[MS-TSTS]:
Terminal Services Terminal Server Runtime Interface Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	0.2	Minor	Made changes to technical and editorial content based on feedback.
10/23/2007	0.3	Minor	Made technical and editorial changes based on feedback.

Date	Revision History	Revision Class	Comments
11/30/2007	0.4	Minor	Made technical and editorial changes based on feedback.
01/25/2008	1.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document specifies the Terminal Services Terminal Server Runtime Interface Protocol. The Terminal Services Terminal Server Runtime Interface Protocol is an RPC-based protocol used for remotely querying and configuring various aspects of a **Terminal Server**.

1.1 Glossary

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

Client
Domain
Endpoint
Globally Unique Identifier (GUID)
Handle
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Opnum
Remote Desktop Protocol (RDP)
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transport
Server
SMB
Terminal Server
Terminal Services
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

GINA: The Graphical Identification and Authentication binary. The binary loaded by **WinLogon** to show the authentication user interface and to validate the user. The default **GINA** (MSGINA) can be replaced by a custom **GINA** if administrators want to use their own authentication UI/method such as fingerprint, voice recognition, and so on. For more information, see [MSDN-GINA].

Listener: A **session** running on a **Terminal Server** and listening for incoming connection requests.

Sessions: A collection of applications simultaneously running under the same Win32 subsystem.

Windows Station (WinStation): **Sessions** running on the computer.

Winlogon: The Windows component that performs interactive logon for a logon **session**.

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

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

1.2.2 Informative References

[MSDN-BCSYSMSG] Microsoft Corporation, "BroadcastSystemMessage", http://msdn2.microsoft.com/en-us/library/ms644932.aspx

[MSDN-GINA] Microsoft Corporation, "Winlogon and GINA", http://msdn2.microsoft.com/en-us/library/aa380543.aspx

[MSDN-MSGBOX] Microsoft Corporation, "Message Box", http://msdn2.microsoft.com/en-us/library/ms645505.aspx

[MSDN-RA] Microsoft Corportation, "Overview of Remote Assistance in Windows XP", May 2004, http://support.microsoft.com/kb/300546

[MSDN-RDP] Microsoft Corporation, "Remote Desktop Protocol", http://msdn2.microsoft.com/En-US/library/aa383015.aspx

[MSDN-RPCBIND] Microsoft Corporation, "RPC Binding and Handles", http://msdn2.microsoft.com/en-us/library/aa373609.aspx

[MSDN-SCWV] Microsoft Corporation, "Service Session Isolation", http://msdn2.microsoft.com/en-us/library/bb203962.aspx

[MSDN-SERIAL] Microsoft Corporation, "Serial Communications in Win32", http://msdn2.microsoft.com/en-us/library/ms810467.aspx

[MSDN-SNDMSG] Microsoft Corporation, "SendMessage Method", http://msdn2.microsoft.com/en-us/library/ms644950.aspx

[MSDN-SOCKET] Microsoft Corporation, "Socket", http://msdn2.microsoft.com/en-us/library/ms740506.aspx

[MSDN-SYSTIME] Microsoft Corporation, "SYSTEMTIME structure", http://msdn2.microsoft.com/en-us/library/ms724950.aspx

[MSDN-TDIADDRESS] Mircrosoft Corporation, "TDI_ADDRESS_IP", http://msdn2.microsoft.com/en-us/library/ms801717.aspx

[MSDN-WLX_CCINFOV20] Microsoft Corporation, "WLX_CLIENT_CREDENTIALS_INFO_V2_0", http://msdn2.microsoft.com/en-us/library/aa381105.aspx

[MSFT-IME] Microsoft Corporation, "What is an IME (Input Method Editor) and how do I use it?", http://www.microsoft.com/globaldev/handson/user/IME_Paper.mspx

[MSFT-WSTSL] Microsoft Corporation, "Windows Server 2003 Terminal Server Licensing", May 2005, http://www.microsoft.com/windowsserver2003/techinfo/overview/termservlic.mspx

[WININTERNALS] Russinovich, M. and Solomon, D., "Microsoft Windows Internals, Fourth Edition", Microsoft Press, 2005, ISBN: 0735619174.http://www.microsoft.com/MSPress/books/6710.aspx

If you have any trouble finding [WININTERNALS], please check here.

1.3 Protocol Overview (Synopsis)

The Terminal Services Terminal Server Runtime Interface Protocol is an RPC-based protocol used for remotely querying and configuring various aspects of a Terminal Server. For example, this protocol can be used to query the number of active **sessions** running on a Terminal Server.

The protocol consists of two major sub-components:

- **Local Session Manager (LSM)**: A central component in the Windows operating system that creates, destroys, and manipulates sessions.
- **TermService**: A specific implementation of connection manager, an extension for the Local Session Manager (LSM). Its main purpose is to service incoming remote connection requests.

The protocol can be further divided into the following functional categories:

- **Local Session Manager (LSM) Session**: These calls deal with collecting information, controlling and configuring sessions running on the Terminal Server.
- **Local Session Manager (LSM) Notification**: These **RPC** calls are asynchronous in nature and can be used to receive event notifications from LSM.
- **Local Session Manager (LSM) Enumeration**: The LSM Enumeration calls are used to enumerate information related to sessions running on a Terminal Server.
- **TermService**: These calls can be used to query and configure various aspects of the TermServices running on the Terminal Server.
- **TermService Listener**: The TermService Listener calls are specific to the **Listener** session running on the Terminal Server and listening for incoming connection requests.
- **Legacy**: All the calls that were used by the TS **clients** in the version of Windows prior to Windows Vista can be grouped under this category.
- **Terminal Server Licensing**: These calls can be used to query information about the licensing mode used by the Terminal Server as well as to configure it remotely.

This is a simple request-response RPC-based protocol. For every method that the **server** receives, it executes the method and returns a completion. The client simply returns the completion status to the caller.

1.4 Relationship to Other Protocols

The Terminal Services Terminal Server Runtime Interface Protocol is dependent upon RPC for its transport. This protocol uses RPC over named pipes as specified in section 2.1.

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

1.5 Prerequisites/Preconditions

The Terminal Services Terminal Server Runtime Interface Protocol is an RPC interface and as a result has the prerequisites specified in [MS-RPCE] as being common to RPC interfaces.

It is assumed that a Terminal Server Runtime Interface (RPC) Protocol client has obtained the name of a Terminal Server that supports the Terminal Server Runtime Interface (RPC) Protocol before this protocol is invoked. How a client does this is outside the scope of this specification.

1.6 Applicability Statement

The Terminal Services Terminal Server Runtime Interface Protocol is only appropriate for querying and configuring a Terminal Server on a Windows operating system.

1.7 Versioning and Capability Negotiation

There are no versioning issues for the Terminal Services Terminal Server Runtime Interface Protocol.

1.8 Vendor-Extensible Fields

The Terminal Services Terminal Server Runtime Interface Protocol uses Win32 error codes. These values are taken from the Windows error number space, as specified in [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meaning.<1> Choosing any other value runs the risk of causing a collision in the future.

1.9 Standards Assignments

In the following table, interfaces are based on section 2.3 of [C706] and named pipes are based on section 2.1.

Description	Interface UUID	Named Pipe
LSM Session (tspubrpc.idl)	{ 484809d6-4239-471b-b5bc-61df8c23ac48 }	\PIPE\ LSM_API_service
LSM Notification (tspubrpc.idl)	{ 11899a43-2b68-4a76-92e3-a3d6ad8c26ce }	\PIPE\ LSM_API_service
LSM Enumeration (tspubrpc.idl)	{ 88143fd0-c28d-4b2b-8fef-8d882f6a9390 }	\PIPE\ LSM_API_service
TermSrv (RCMPublic.idl)	{ bde95fdf-eee0-45de-9e12-	\PIPE\ TermSrv_API_service

Description	Interface UUID	Named Pipe
	e5a61cd0d4fe }	
TermSrv Listener (RCMPublic.idl)	{ 497d95a6-2d27-4bf5-9bbd-a6046957133c }	\PIPE\ TermSrv_API_service
Legacy (Legacy.idl)	{ 5ca4a760-ebb1-11cf-8611-00a0245420ed }	\PIPE\ Ctx_WinStation_API_service
Terminal Service Licensing (Lcrpc.idl)	{ 2f59a331-bf7d-48cb-9e5c-7c090d76e8b8 }	\LcRpc

2 Messages

The following sections specify how Terminal Services Terminal Server Runtime Interface Protocol messages are transported, message syntax, and common data types.

2.1 Transport

The Terminal Services Terminal Server Runtime Interface Protocol uses the **RPC protocol sequences** as specified in section 2.11 of [MS-RPCE] and section 1.4 of [MS-RPCH].

The Terminal Services Terminal Server Runtime Interface Protocol uses the following static **endpoints** as well as **well-known endpoints**. These endpoints are ports for section 1.5 of [MS-RPCH] and section 2.11 of [MS-RPCE] on the TS server.

- Port 443: This endpoint is used by the RPC Over HTTP Protocol as the underlying transport.
- Port 3389: This endpoint is used by the TS 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 TS client and target server.

Both endpoints MUST be supported.

The TS protocol MUST use the **UUID** as specified in section 1.9. The RPC version number is 1.0.

2.2 Common Data Types

In addition to RPC base types specified in [C706] and [MS-RPCE], this document uses the following definitions, as specified in [MS-DTYP]:

- **BOOL**
- **BOOLEAN**
- **BYTE**
- **CHAR**
- **DWORD**
- **HANDLE**
- **HRESULT**
- **LONG**
- **UCHAR**
- **UINT**
- **ULONG**
- **USHORT**
- **VOID**
- **WCHAR**

The data types in the following sections are defined in the **Microsoft Interface Definition Language (MIDL)** specification for this RPC interface, as specified in section 6.

2.2.1 Data Types

2.2.1.1 SESSION_HANDLE

This type is declared as follows:

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

A **handle** to a session on the Terminal Server. Returned by **RpcOpenSession**.

2.2.1.2 SESSION_HANDLE_EXCLUSIVE

TBD

An exclusive handle to a session on the Terminal Server.

This type is declared as follows:

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

2.2.1.3 ENUM_HANDLE

This type is declared as follows:

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

A handle representing the session enumeration object on the Terminal Server. Returned by **sRpcOpenEnum**.

2.2.1.4 HLISTENER

This type is declared as follows:

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

A handle representing a listener running on the Terminal Server.

2.2.1.5 HLISTENER_EXCLUSIVE

This type is declared as follows:

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

An exclusive handle representing a listener running on the Terminal Server.

2.2.1.6 SERVER_HANDLE

This type is declared as follows:

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

A handle returned by **RpcWinStationOpenServer** on the **Terminal Services** server.

2.2.1.7 SERVER_HANDLE_EXCLUSIVE

An exclusive handle to the Terminal Services server.

This type is declared as follows:

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

2.2.1.8 WINSTATIONNAME

An array of **WCHAR** (WINSTATIONNAMEW) or **CHAR** (WINSTATIONNAMEA) characters that represent the name of a session. For example, "Rdp-Tcp#3" for the third **RDP** over TCP session created.

```
typedef WCHAR WINSTATIONNAMEW[WINSTATIONNAME_LENGTH + 1];

typedef CHAR WINSTATIONNAMEA[WINSTATIONNAME_LENGTH + 1];
```

2.2.1.9 DLLNAME

The name of a DLL.

```
typedef WCHAR DLLNAMEW[DLLNAME_LENGTH + 1];

typedef WCHAR* PDLLNAMEW;

typedef CHAR DLLNAMEA[DLLNAME_LENGTH + 1];

typedef CHAR* PDLLNAMEA;
```

The following block determines **DLLNAME**:

```
#ifdef UNICODE
#define DLLNAME DLLNAMEW
#define PDLLNAME PDLLNAMEW
#else
#define DLLNAME DLLNAMEA
#define PDLLNAME PDLLNAMEA
#endif /* UNICODE */
```

2.2.1.10 DEVICENAME

The name of a device.

```
typedef WCHAR DEVICENAMEW[DEVICENAME_LENGTH + 1];

typedef WCHAR* PDEVICENAMEW;

typedef CHAR DEVICENAMEA[DEVICENAME_LENGTH + 1];

typedef CHAR* PDEVICENAMEA;
```

The folloowing block determines **DEVICENAME**:

```
#ifdef UNICODE
#define DEVICENAME DEVICENAMEW
#define PDEVICENAME PDEVICENAMEW
#else
#define DEVICENAME DEVICENAMEA
#define PDEVICENAME PDEVICENAMEA
#endif /* UNICODE */
```

2.2.1.11 VIRTUALCHANNELNAME

The name of a virtual channel.

```
typedef CHAR VIRTUALCHANNELNAME[ VIRTUALCHANNELNAME LENGTH + 1 ];  // includes
typedef CHAR * PVIRTUALCHANNELNAME;
```

2.2.1.12 WINSTATIONINFOCLASS

The **WINSTATIONINFOCLASS** enumeration is used by **RpcWinStationQueryInformation** and **RpcWinStationSetInformation** to indicate the class of data to either query for or set on the server. In the following section, a brief description of each info class is appended to each enum value. See **RpcWinStationQueryInformation** for information on classes of data that can be queried and **RpcWinStationSetInformation** for classes of data that can be set. The enum value **WinStationUnused1** is not currently used.<2>

```
typedef  enum _WINSTATIONINFOCLASS
{
  WinStationCreateData,
  WinStationConfiguration,
  WinStationPdParams,
  WinStationWd,
  WinStationPd,
  WinStationPrinter,
  WinStationClient,
  WinStationModules,
  WinStationInformation,
  WinStationTrace,
  WinStationBeep,
  WinStationEncryptionOff,
  WinStationEncryptionPerm,
  WinStationNtSecurity,
  WinStationUserToken,
  WinStationUnused1,
  WinStationVideoData,
  WinStationInitialProgram,
  WinStationCd,
  WinStationSystemTrace,
  WinStationVirtualData,
  WinStationClientData,
  WinStationSecureDesktopEnter,
  WinStationSecureDesktopExit,
  WinStationLoadBalanceSessionTarget,
  WinStationLoadIndicator,
  WinStationShadowInfo,
  WinStationDigProductId,
  WinStationLockedState,
  WinStationRemoteAddress,
  WinStationIdleTime,
  WinStationLastReconnectType,
  WinStationDisallowAutoReconnect,
  WinStationMprNotifyInfo,
  WinStationExecSrvSystemPipe,
  WinStationSmartCardAutoLogon,
  WinStationIsAdminLoggedOn,
  WinStationReconnectedFromId,
  WinStationEffectsPolicy
} WINSTATIONINFOCLASS;
```

WinStationCreateData: Query the data used to create an instance of **WinStation**.

WinStationConfiguration: Query or set WinStation parameters.

WinStationPdParams: Query or set PD (Protocol Drivers) parameters.

WinStationWd: Query WD (Window Driver) configuration (only one can be loaded).

WinStationPd: Query PD configuration (many can be loaded).

WinStationPrinter: Query or set LPT (Line Printer Terminal) mapping to printer queues.

WinStationClient: Query information about client.

WinStationModules: Query information about all client modules.

WinStationInformation: Query information about WinStation.

WinStationTrace: Enable or disable WinStation tracing.

WinStationBeep: Sound a beep in the WinStation.

WinStationEncryptionOff: Turn off encryption.

WinStationEncryptionPerm: Encryption is permanently on.

WinStationNtSecurity: Select **Winlogon** security desktop.

WinStationUserToken: Query primary access token of the logged-on user.

WinStationUnused1: Not used.

WinStationVideoData: Query horizontal resolution, vertical resolution, and color depth.

WinStationInitialProgram: Identify Initial Program that Terminal Services runs when the user logs on.

WinStationCd: Query CD (Client Device) config (only one can be loaded).

WinStationSystemTrace: Enable or disable system tracing.

WinStationVirtualData: Query client virtual data.

WinStationClientData: Send data to a client.

WinStationSecureDesktopEnter: Turn encryption on, if enabled.

WinStationSecureDesktopExit: Turn encryption off, if enabled.

WinStationLoadBalanceSessionTarget: Load balance information from redirected client.

WinStationLoadIndicator: Query load capacity information.

WinStationShadowInfo: Query or set Shadow state and parameters.

WinStationDigProductId: Get **WINSTATIONPRODID**, as specified in section 2.2.2.31.

WinStationLockedState: Used by winlogon to notify applications and services.

WinStationRemoteAddress: Query client IP address.

WinStationIdleTime: Query for how much time the WinStation is idle.

WinStationLastReconnectType: Query if last reconnect for this WinStation was manual or auto-reconnect.

WinStationDisallowAutoReconnect: Allow or disallow auto-reconnect for this WinStation.

WinStationMprNotifyInfo: Used to retrieve user credential from winlogon, which can then be used to notify network providers.

WinStationExecSrvSystemPipe: Retrieves the name of the pipe that TermSrv listens on and that can be used to launch a process inside a session from another session. Only local service, network service, and local system can open/write to this pipe.<3>

WinStationSmartCardAutoLogon: Indicates a smart card auto-logon.

WinStationIsAdminLoggedOn: Indicates whether the currently logged on user is an administrator.

WinStationReconnectedFromId: In case of reconnected sessions, this will return the session ID of the temporary session from which it was reconnected, or -1 if no temporary session was created.

WinStationEffectsPolicy: Return policies that differentiate between different versions of Windows.

2.2.1.13 WINSTATIONSTATECLASS

The **WINSTATIONSTATECLASS** enumeration represents the current state of a session.

```
typedef  enum _WINSTATIONSTATECLASS
{
  State_Active = 0,
  State_Connected = 1,
  State_ConnectQuery = 2,
  State_Shadow = 3,
  State_Disconnected = 4,
  State_Idle = 5,
  State_Listen = 6,
  State_Reset = 7,
  State_Down = 8,
  State_Init = 9
} WINSTATIONSTATECLASS;
```

State_Active: A user is logged on to a session and the client is connected.

State_Connected: A client is connected to a session but the user has not logged on yet.

State_ConnectQuery: A session is in the process of connecting to a client.

State_Shadow: A session is shadowing another session.

State_Disconnected: A user is logged on to the session but the client is currently disconnected from the server.

State_Idle: A session is waiting on a client to connect to the server.

State_Listen: A listener waiting for connections from TS client.

State_Reset: A session is being reset which will lead to the user being logged off, the session being terminated, and the client disconnected.<4>

State_Down: A session is currently tearing down or is in the down state, indicating an error.

State_Init: A session is in the process of being initialized.

2.2.1.14 SDCLASS

The **SDCLASS** (stack driver class) enumeration is used to specify a type of binary or driver in the union PDPARAMS and to indicate which structure in the union PDPARAMS actually applies to a given instance of the PDPARAMS structure. In the most common use of PDPARAMS on Windows and the TS RDP implementation in Windows, only SdNetwork and SdAsync are actually used.

```
typedef  enum _SDCLASS
{
  SdNone = 0,
  SdConsole = 1,
  SdNetwork = 2,
  SdAsync = 3,
  SdOemTransport = 4,
  SdISDN = 5,
  SdX25 = 6,
  SdModem = 7,
  SdOemConnect = 8,
  SdFrame = 9,
  SdReliable = 10,
  SdEncrypt = 11,
  SdCompress = 12,
  SdTelnet = 13,
  SdOemFilter = 14,
  SdNasi = 15,
  SdTapi = 16,
  SdReserved1 = 17,
  SdReserved2 = 18,
  SdReserved3 = 19,
  SdClass_Maximum = 20
} SDCLASS;
```

SdNone: None.

SdConsole: No DLL.

SdNetwork: Indicates the following DLLs: tdnetb.dll, tdspx.dll, tdftp.dll tdipx.dll.

SdAsync: Indicates tdasync.dll.

SdOemTransport: User transport driver.

SdISDN: Not implemented.

SdX25: Not implemented.

SdModem: Indicates pdmodem.dll.

SdOemConnect: User protocol driver.

SdFrame: Indicates pdframe.dll.

SdReliable: Indicates pdreli.dll.

SdEncrypt: Indicates pdcrypt1.dll.

SdCompress: Indicates pdcomp.dll.

SdTelnet: Not implemented.

SdOemFilter: User protocol driver.

SdNasi: Indicates tdnasi.dll.

SdTapi: Indicates pdtapi.dll.

SdReserved1: Reserved value.

SdReserved2: Reserved value.

SdReserved3: Reserved value.

SdClass_Maximum: MUST be last.

2.2.1.15 SHADOWCLASS

The **SHADOWCLASS** enumeration is used to indicate the shadow-related settings for a session running on a Terminal Server.

```
typedef  enum _SHADOWCLASS
{
  Shadow_Disable = 0,
  Shadow_EnableInputNotify = 1,
  Shadow_EnableInputNoNotify = 2,
  Shadow_EnableNoInputNotify = 3,
  Shadow_EnableNoInputNoNotify = 4
} SHADOWCLASS;
```

Shadow_Disable: Shadowing is disabled.

Shadow_EnableInputNotify: Permission is asked first from the session being shadowed. The shadower is also permitted keyboard and mouse input.

Shadow_EnableInputNoNotify: Permission is not asked first from the session being shadowed. The shadower is also permitted keyboard and mouse input.

Shadow_EnableNoInputNotify: Permission is asked first from the session being shadowed. The shadower is not permitted keyboard and mouse input and MUST observe the shadowed session.

Shadow_EnableNoInputNoNotify: Permission is not asked first from the session being shadowed. The shadower is not permitted keyboard and mouse input and MUST observe the shadowed session.

2.2.1.16 RECONNECT_TYPE

The **RECONNECT_TYPE** enumeration specifies the reconnect type of the last session reconnect.

```
typedef  enum _RECONNECT_TYPE
{
```

```
  NeverReconnected = 0,
  ManualReconnect = 1,
  AutoReconnect = 2
} RECONNECT_TYPE,
 *PRECONNECT_TYPE;
```

NeverReconnected: Session has never been reconnected to. This is the default type until the first time the session has been reconnected to.

ManualReconnect: Session was disconnected from and was manually reconnected to by the user.

AutoReconnect: Session was disconnected from and was automatically reconnected to by the TS client and the server negotiating the reconnect without input from the user.

2.2.1.17 CLIENTDATANAME

Specifies the name of the client data being provided.

```
typedef CHAR CLIENTDATANAME[CLIENTDATANAME_LENGTH + 1];

typedef CHAR* PCLIENTDATANAME;
```

The name has the following form:

```
name syntax:  xxxyyyy<null>
```

The elements are as follows:

xxx: The OEM ID.

yyyy: Client data name.

<null>: Trailing null.

Currently the only three values defined for this data type are the following:

Value	Meaning
CLIENTDATA_SERVER	Indicates that the data being provided is the server's name.
CLIENTDATA_USERNAME	Indicates that the data is the user's username.
CLIENTDATA_DOMAIN	Indicates that the data being provided is the user's **domain** name.

2.2.1.18 LCRPC_HANDLE

Handle to the Terminal Server used to query or configure the licensing policies of the server.

This type is declared as follows:

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

2.2.1.19 LCRPC_HANDLE_EXCLUSIVE

Handle to the Terminal Server used to query or configure the licensing policies of the server. Access to this handle is serialized.

This type is declared as follows:

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

2.2.1.20 TNotificationId

Specifies the type of notification to wait for from the Terminal Server.

This type is declared as follows:

```
typedef ULONG TNotificationId;
```

It MUST be a bitwise OR of any of the following values:

Value	Description
WTS_NOTIFY_NONE 0	No Notification
WTS_NOTIFY_CREATE 0x1	Session Creation Notification
WTS_NOTIFY_CONNECT 0x2	Session Connection Notification
WTS_NOTIFY_DISCONNECT 0x4	Session Disconnection Notification
WTS_NOTIFY_LOGON 0x8	Session Logon Notification
WTS_NOTIFY_LOGOFF 0x10	Session Logoff Notification
WTS_NOTIFY_SHADOW_START	Session Shadow Start Notification

Value	Description
0x20	
WTS_NOTIFY_SHADOW_STOP 0x40	Session Shadow Stop Notification
WTS_NOTIFY_TERMINATE 0x80	Session Termination Notification
WTS_NOTIFY_CONSOLE_CONNECT 0x100	Console Session Connection Notification
WTS_NOTIFY_CONSOLE_DISCONNECT 0x200	Console Session Disconnect Notification
WTS_NOTIFY_LOCK 0x400	Session Lock Notification
WTS_NOTIFY_UNLOCK 0x800	Session Unlock Notification
WTS_NOTIFY_ALL 0xffffffff	All Notifications

2.2.1.21 NOTIFY_HANDLE

A handle to a notification object.

A handle to a notification object. Used in asynchronous calls such as **RpcRegisterAsyncNotification** and **RpcWaitAsyncNotification**.

This type is declared as follows:

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

2.2.1.22 NOTIFY_HANDLE_EXCLUSIVE

An exclusive handle to a notification object.

This type is declared as follows:

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

2.2.1.23 SIZE_T

A data type equivalent to **size_t**.

This type is declared as follows:

```
typedef ULONG_PTR SIZE_T;
```

2.2.1.24 BOUNDED_ULONG

A bounded **ULONG**.

This type is declared as follows:

```
typedef [range(0, 0x8000)] ULONG BOUNDED_ULONG;
```

2.2.1.25 UINT_PTR

A pointer to an unsigned integer, whose length is dependent on processor word size.

```
#if  defined(_WIN64)
typedef unsigned __int64 UINT_PTR;
#else
typedef unsigned int UINT_PTR;
#endif
```

2.2.1.26 REGISTRATION_HANDLE

A registration handle returned by **RpcWinStationRegisterNotificationEvent**.

This type is declared as follows:

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

2.2.2 Structures

2.2.2.1 SESSION_FILTER

The **SESSION_FILTER** enumeration specifies the types of filters to apply when retrieving the list of session IDs running on a Terminal Server. Currently, there is only one type of filter exposed by way of RPC.

```
typedef  enum _SESSION_FILTER
{
  SF_SERVICES_SESSION_POPUP = 0
} SESSION_FILTER;
```

SF_SERVICES_SESSION_POPUP: Returns all sessions in logged-on state.

2.2.2.2 PROTOCOLSTATUS_INFO_TYPE

The **PROTOCOLSTATUS_INFO_TYPE** enumeration specifies the protocol status information requested for a particular session running on a Terminal Server.

```
typedef  enum
{
  PROTOCOLSTATUS_INFO_BASIC = 0,
  PROTOCOLSTATUS_INFO_EXTENDED = 1
} PROTOCOLSTATUS_INFO_TYPE;
```

PROTOCOLSTATUS_INFO_BASIC: Returns basic information about the protocol status in a **PROTOCOLSTATUS** structure.

PROTOCOLSTATUS_INFO_EXTENDED: Returns extended information about the protocol status. Extended information is returned in a **PROTOCOLSTATUS** structure.

2.2.2.3 QUERY_SESSION_DATA_TYPE

The **QUERY_SESSION_DATA_TYPE** enumeration specifies the type of session information that can be requested for a particular session running on a Terminal Server (TS).

```
typedef  enum
{
  QUERY_SESSION_DATA_MODULE = 0,
  QUERY_SESSION_DATA_WDCONFIG,
  QUERY_SESSION_DATA_VIRTUALDATA,
  QUERY_SESSION_DATA_LICENSE,
  QUERY_SESSION_DATA_DEVICEID
} QUERY_SESSION_DATA_TYPE;
```

QUERY_SESSION_DATA_MODULE: Used to retrieve data about protocol-specific binaries loaded for the given TS session.

QUERY_SESSION_DATA_WDCONFIG: Used to retrieve protocol driver configuration data for the session.

QUERY_SESSION_DATA_VIRTUALDATA: Used to retrieve data about virtual channels for the given TS session.

QUERY_SESSION_DATA_LICENSE: Used to retrieve data about type of license associated with a given TS session.

QUERY_SESSION_DATA_DEVICEID: Used to retrieve the device ID of the client connected to a given TS session.<5>

2.2.2.4 PSESSIONENUM

PSESSIONENUM is a pointer to a structure containing information about the sessions running on the Terminal Server. Returned by **RpcGetEnumResult**.

```
typedef struct _SESSIONENUM {
  DWORD Level;
  [switch_is(Level)] SessionInfo Data;
} SESSIONENUM,
 *PSESSIONENUM;
```

Level: The level of information contained in Data, current valid value are 1, 2, and 3.

Data: Contains information at a specified level of detail about sessions running on a computer.

2.2.2.4.1 SessionInfo

The **SessionInfo** is a union of structures, each structure capable of holding information about sessions running on a computer at a different level of detail, as specified in sections 2.2.2.4.1.1, 2.2.2.4.1.2, and 2.2.2.4.1.3, respectively.

```
typedef
[switch_type(DWORD)]
union _SessionInfo {
  [case(1)]
    SESSIONENUM_LEVEL1 SessionEnum_Level1;
  [case(2)]
    SESSIONENUM_LEVEL2 SessionEnum_Level2;
  [case(3)]
    SESSIONENUM_LEVEL3 SessionEnum_Level3;
  [default]
    ;
} SessionInfo;
```

SessionEnum_Level1: A **SESSIONENUM_LEVEL1** structure containing a level of information about sessions running on a computer.

SessionEnum_Level2: A **SESSIONENUM_LEVEL2** structure containing a level of information about sessions running on a computer.

SessionEnum_Level3: A **SESSIONENUM_LEVEL3** structure containing a level of information about sessions running on a computer.

2.2.2.4.1.1 SESSIONENUM_LEVEL1

The **SESSIONENUM_LEVEL1** structure contains basic information about sessions running on a computer.

```
typedef struct _SESSIONENUM_LEVEL1 {
  LONG SessionId;
  LONG State;
  WCHAR Name[33];
} SESSIONENUM_LEVEL1,
 *PSESSIONENUM_LEVEL1;
```

SessionId: An identifier assigned to the session contained in this structure by the operating system.

State: The state the session is in, as specified in section 3.2.4.1.8.

Name: A null-terminated string containing the name of the session assigned by Terminal Services.

2.2.2.4.1.2 SESSIONENUM_LEVEL2

The **SESSIONENUM_LEVEL2** structure contains information about sessions running on a computer with more details than **SESSIONENUM_LEVEL1**.

```
typedef struct _SESSIONENUM_LEVEL2 {
  LONG SessionId;
  LONG State;
  WCHAR Name[33];
  ULONG Source;
  BOOL bFullDesktop;
  GUID SessionType;
  GUID LicenseType;
} SESSIONENUM_LEVEL2,
 *PSESSIONENUM_LEVEL2;
```

SessionId: An identifier assigned to the session contained in this structure by the operating system.

State: The state the session is in, as specified in section 3.2.4.1.8.

Name: The name assigned to the session by the operating system.

Source: Defines the type of device that connected to this particular session on the Terminal Server. It MUST be one of the following values.

Value	Meaning
1	Connected from a computer running a Windows client or server operating system.
2	Connected from a device running Media Center Edition. Currently only Xbox is using this value.

Value	Meaning
3	Service Terminal

bFullDesktop: Specifies whether the session is for a complete desktop.

SessionType: Describes the type of the session. It MUST have the following values based on what terminal was used to connect to the particular session.

Value	Meaning
TERMINAL_TYPE_SERVICE 88f5767d-d13f-404d-a348-8b8e030294a9	GUID for Service Terminal
TERMINAL_TYPE_REGULAR_DESKTOP 0f0a4bf8-8362-435d-938c-222a518a8b78	GUID for Regular Desktop Remote Terminal
TERMINAL_TYPE_RDP_REMOTEAP eddcc3ce-6e7e-4f4b-8439-3d9ad4c9440f	GUID for Remote Applications Terminal
TERMINAL_TYPE_MCE 8dc86f1d-9969-4379-91c1-06fe1dc60575	GUID for Media Center Edition Terminal

LicenseType: Describes the type of license that a session consumes. It MUST be one of the following values.

Value	Meaning
LICENSE_TYPE_DEFAULT 00000000-0000-0000-0000-000000000000	GUID for default license
LICENSE_TYPE_BUILTIN 45344fe7-00e6-4ac6-9f01-d01fd4ffadfb	GUID for built-in license
LICENSE_TYPE_APPSERVER 36e64007-aef1-4085-89e3-66c26abade84	GUID for the TS Application Server license
LICENSE_TYPE_MCE 8dc86f1d-9969-4379-91c1-06fe1dc60575	GUID for Media Center License

2.2.2.4.1.3 SESSIONENUM_LEVEL3

The **SESSIONENUM_LEVEL3** structure contains information about sessions running on a computer with more details than **SESSIONENUM_LEVEL1** and **SESSIONENUM_LEVEL2**.

```
typedef struct _SESSIONENUM_LEVEL3 {
  LONG SessionId;
  LONG State;
  WCHAR Name[33];
  ULONG Source;
  BOOL bFullDesktop;
  GUID SessionType;
  GUID LicenseType;
  ULONG ProtoDataSize;
  [size_is(ProtoDataSize)] UCHAR* pProtocolData;
} SESSIONENUM_LEVEL3,
```

```
*PSESSIONENUM_LEVEL3;
```

SessionId: An identifier assigned to the session contained in this structure by the operating system.

State: The state the session is in, as specified in section 3.2.4.1.8.

Name: The name assigned to the session by the operating system.

Source: Defines the type of device that connected to this particular session on the Terminal Server. It can have the following values.

Value	Meaning
1	Connected from a computer running a Windows client or server operating system.
2	Connected from a device running Media Center Edition. Currently only Xbox is using this.
3	Service Terminal

bFullDesktop: Specifies whether the session is for a complete desktop.

SessionType: Describes the type of the session. It MUST have the following values based on what terminal was used to connect to the particular session.

Value	Meaning
TERMINAL_TYPE_SERVICE 88f5767d-d13f-404d-a348-8b8e030294a9	GUID for Service Terminal
TERMINAL_TYPE_REGULAR_DESKTOP 0f0a4bf8-8362-435d-938c-222a518a8b78	GUID for Regular Desktop Remote Terminal
TERMINAL_TYPE_RDP_REMOTEAP eddcc3ce-6e7e-4f4b-8439-3d9ad4c9440f	GUID for Remote Applications Terminal
TERMINAL_TYPE_MCE 8dc86f1d-9969-4379-91c1-06fe1dc60575	GUID for Media Center Edition Terminal

LicenseType: Describes the type of license that a session consumes. It MUST be one of the following values.

Value	Meaning
LICENSE_TYPE_DEFAULT 00000000-0000-0000-0000-000000000000	GUID for default license
LICENSE_TYPE_BUILTIN 45344fe7-00e6-4ac6-9f01-d01fd4ffadfb	GUID for built-in license
LICENSE_TYPE_APPSERVER 36e64007-aef1-4085-89e3-66c26abade84	GUID for the TS Application Server license
LICENSE_TYPE_MCE	GUID for Media Center License

Value	Meaning
8dc86f1d-9969-4379-91c1-06fe1dc60575	

ProtoDataSize: Size of data, in bytes, contained in *pProtocolData*.

pProtocolData: Data containing information about the protocol status between the Terminal Server Client and Server.

2.2.2.5 PLSMSESSIONINFORMATION

PLSMSESSIONINFORMATION is a pointer to a structure that contains information about a session running on a Terminal Server.

```
typedef struct _LSMSessionInformation {
  [string] WCHAR* pszUserName;
  [string] WCHAR* pszDomain;
  [string] WCHAR* pszTerminalName;
  LONG SessionState;
  BOOL DesktopLocked;
  hyper ConnectTime;
  hyper DisconnectTime;
  hyper LogonTime;
} LSMSESSIONINFORMATION,
 *PLSMSESSIONINFORMATION;
```

pszUserName: The name of the user logged on to the session.

pszDomain: The domain to which the currently logged-in user belongs.

pszTerminalName: The name of the terminal associated with the specific session.

SessionState: The state of the session, as described in section 3.2.4.1.8.

DesktopLocked: Set to TRUE if session is currently locked, or FALSE otherwise.

ConnectTime: The time the session was last connected to.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

DisconnectTime: The time the session was last disconnected from.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

LogonTime: The time the session was last logged on to.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

2.2.2.6 PLISTENERENUM

PLISTENERENUM contains information about one Terminal Server listener and the level of detail of the information provided.

```
typedef struct _LISTENERENUM {
  DWORD Level;
```

```
  [switch_is(Level)] ListenerInfo Data;
} LISTENERENUM,
 *PLISTENERENUM;
```

Level: The level of detail provided about the listener. The only supported value is 1.

Data: Information about the listener. This is of the type **ListenerInfo**.

2.2.2.6.1 ListenerInfo

ListenerInfo is a union of structures, each member containing a different level of information about a Terminal Server listener.

```
typedef
[switch_type(DWORD)]
union _ListenerInfo {
  [case(1)]
    LISTENERENUM_LEVEL1 ListenerEnum_Level1;
  [default]
    ;
} ListenerInfo,
 *PListenerInfo;
```

ListenerEnum_Level1: The only supported member of the union. It contains listener information of level 1. It is of the type **LISTENERENUM_LEVEL1**.

2.2.2.6.1.1 LISTENERENUM_LEVEL1

LISTENERENUM_LEVEL1 is a structure containing information of level 1 detail about a TS listener.

```
typedef struct _LISTENERENUM_LEVEL1 {
  LONG Id;
  BOOL bListening;
  WCHAR Name[33];
} LISTENERENUM_LEVEL1,
 *PLISTENERENUM_LEVEL1;
```

Id: The identifier associated with the listener.

bListening: Set to TRUE if the TS listener is listening for incoming connections, or FALSE otherwise.

Name: A null-terminated string containing the name of the listener.

2.2.2.7 LOGONID

LOGONID is a macro defined to be the structure SESSIONID. This type represents information about the session or WinStation identified by the identifier SessionId. For more information, see the macro definition in section 6.5.

```
typedef struct _SESSIONIDW {
  union {
    ULONG SessionId;
    ULONG LogonId;
  } _SessionId_LogonId_union;
  WINSTATIONNAMEW WinStationName;
  WINSTATIONSTATECLASS State;
} SESSIONIDW,
 *PSESSIONIDW;
```

SessionId: In a union with LogonId. Represents WinStation or session identifier numbered from 0-65535 for TS sessions. If the number is 65536 or higher, then the WinStation is actually a listening WinStation.

LogonId: In a union with **SessionId**, is only used internally within TS code.

WinStationName: The name of the WinStation represented by this structure. See section 2.2.1.8 for more information on the type WINSTATIONNAME.

State: The current state of the WinStation. See section 2.2.1.13 for more information on the type WINSTATIONSTATECLASS.

2.2.2.8 _TS_PROCESS_INFORMATION_NT4

The **_TS_PROCESS_INFORMATION_NT4** structure is returned by **RpcWinStationEnumerateProcesses**.

```
typedef struct __TS_PROCESS_INFORMATION {
  ULONG MagicNumber;
  ULONG LogonId;
  void* ProcessSid;
  ULONG Pad;
} _TS_PROCESS_INFORMATION_NT4,
 *_TS_PROCESS_INFORMATION_NT4;
```

MagicNumber: MUST be set to TS_PROCESS_INFO_MAGIC_NT4 (0x23495452).

LogonId: The session ID of the process.

ProcessSid: The security identifier (SID) of the owner of the process.

Pad: MUST be set to 0.

2.2.2.9 TS_ALL_PROCESSES_INFO

TS_ALL_PROCESSES_INFO contains data on all the processes on the system accessible to the user who issued the call.

```
typedef struct _TS_ALL_PROCESSES_INFO {
  PTS_SYS_PROCESS_INFORMATION pTsProcessInfo;
  DWORD SizeOfSid;
  BYTE* pSid;
```

```
} TS_ALL_PROCESSES_INFO,
 *PTS_ALL_PROCESSES_INFO;
```

pTsProcessInfo: Pointer to the process information **TS_SYS_PROCESS_INFORMATION**.

SizeOfSid: Size of **pSid**.

pSid: **SID** (security identifier), as specified in [MS-DTYP], of the owner of the process.

2.2.2.9.1 TS_SYS_PROCESS_INFORMATION

The **TS_SYS_PROCESS_INFORMATION** structure contains information about a process running on a system.

```
typedef struct _TS_SYS_PROCESS_INFORMATION {
  ULONG NextEntryOffset;
  ULONG NumberOfThreads;
  LARGE_INTEGER SpareLi1;
  LARGE_INTEGER SpareLi2;
  LARGE_INTEGER SpareLi3;
  LARGE_INTEGER CreateTime;
  LARGE_INTEGER UserTime;
  LARGE_INTEGER KernelTime;
  TS_UNICODE_STRING ImageName;
  LONG BasePriority;
  DWORD UniqueProcessId;
  DWORD InheritedFromUniqueProcessId;
  ULONG HandleCount;
  ULONG SessionId;
  ULONG SpareUl3;
  SIZE_T PeakVirtualSize;
  SIZE_T VirtualSize;
  ULONG PageFaultCount;
  ULONG PeakWorkingSetSize;
  ULONG WorkingSetSize;
  SIZE_T QuotaPeakPagedPoolUsage;
  SIZE_T QuotaPagedPoolUsage;
  SIZE_T QuotaPeakNonPagedPoolUsage;
  SIZE_T QuotaNonPagedPoolUsage;
  SIZE_T PagefileUsage;
  SIZE_T PeakPagefileUsage;
  SIZE_T PrivatePageCount;
} TS_SYS_PROCESS_INFORMATION,
 *PTS_SYS_PROCESS_INFORMATION;
```

NextEntryOffset: Offset to the start of data for the next process.

NumberOfThreads: Number of threads in the process.

SpareLi1: Reserved.

SpareLi2: Reserved.

SpareLi3: Reserved.

CreateTime: Creation time of the process.

UserTime: Amount of time the process has spent running in user mode.

KernelTime: Amount of time the process has spent running in kernel mode.

ImageName: String containing the process's image name.

BasePriority: Base priority of the process.

UniqueProcessId: Process's unique process ID.

InheritedFromUniqueProcessId: Parent process's unique process ID.

HandleCount: Current number of handles open in the process.

SessionId: Session identifier of the session the process is in.

SpareUl3: Reserved.

PeakVirtualSize: Peak size of virtual memory used by the process.

VirtualSize: Current size of virtual memory used by the process.

PageFaultCount: Number of page faults in the process.

PeakWorkingSetSize: Peak size of the working set of the process.

WorkingSetSize: Current size of working set of the process.

QuotaPeakPagedPoolUsage: Peak quota charged to the process for paged pool usage.

QuotaPagedPoolUsage: Current quota charged to the process for paged pool usage.

QuotaPeakNonPagedPoolUsage: Peak quota charged to the process for nonpaged pool usage.

QuotaNonPagedPoolUsage: Current quota charged to the process for nonpaged pool usage.

PagefileUsage: Amount of bytes of page file storage in use by the process.

PeakPagefileUsage: Peak amount of bytes of page file storage in use by the process.

PrivatePageCount: Current number of memory pages allocated by the process.

2.2.2.9.1.1 TS_UNICODE_STRING

A UNICODE string.

```
typedef struct _TS_UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  [size_is(MaximumLength / 2), length_is(Length / 2)]
    PWSTR Buffer;
} TS_UNICODE_STRING;
```

Length: The actual length of the string currently stored in **Buffer**, in bytes.

MaximumLength: The maximum length of the string that could be stored in **Buffer**, in bytes.

Buffer: A wide character string. May not be NULL-terminated.

2.2.2.10 TS_ALL_PROCESSES_INFO_NT6

The **TS_ALL_PROCESSES_INFO_NT6** structure contains data on all the processes on the system accessible using the user's credentials.

```
typedef struct _TS_ALL_PROCESSES_INFO_NT6 {
  PTS_SYS_PROCESS_INFORMATION_NT6 pTsProcessInfo;
  DWORD SizeOfSid;
  [size_is(SizeOfSid)] PBYTE pSid;
} TS_ALL_PROCESSES_INFO_NT6,
 *PTS_ALL_PROCESSES_INFO_NT6;
```

pTsProcessInfo: Pointer to the process information.

SizeOfSid: Size, in bytes, of Sid structure pointed to by **pSid**.

pSid: SID (security identifier) of the process.

2.2.2.10.1 TS_SYS_PROCESS_INFORMATION_NT6

The **TS_SYS_PROCESS_INFORMATION_NT6** structure contains information about a process running on a system.

```
typedef struct _TS_SYS_PROCESS_INFORMATION_NT6 {
  ULONG NextEntryOffset;
  ULONG NumberOfThreads;
  LARGE_INTEGER SpareLi1;
  LARGE_INTEGER SpareLi2;
  LARGE_INTEGER SpareLi3;
  LARGE_INTEGER CreateTime;
  LARGE_INTEGER UserTime;
  LARGE_INTEGER KernelTime;
  NT6_TS_UNICODE_STRING ImageName;
  LONG BasePriority;
  DWORD UniqueProcessId;
  DWORD InheritedFromUniqueProcessId;
  ULONG HandleCount;
  ULONG SessionId;
  ULONG SpareUl3;
  SIZE_T PeakVirtualSize;
  SIZE_T VirtualSize;
  ULONG PageFaultCount;
  ULONG PeakWorkingSetSize;
  ULONG WorkingSetSize;
  SIZE_T QuotaPeakPagedPoolUsage;
  SIZE_T QuotaPagedPoolUsage;
  SIZE_T QuotaPeakNonPagedPoolUsage;
  SIZE_T QuotaNonPagedPoolUsage;
  SIZE_T PagefileUsage;
```

```
  SIZE_T PeakPagefileUsage;
  SIZE_T PrivatePageCount;
} TS_SYS_PROCESS_INFORMATION_NT6,
 *PTS_SYS_PROCESS_INFORMATION_NT6;
```

NextEntryOffset: Offset to the start of data for the next process.

NumberOfThreads: Number of threads in the process.

SpareLi1: Reserved.

SpareLi2: Reserved.

SpareLi3: Reserved.

CreateTime: Creation time of the process.

UserTime: Amount of time the process has spent running in user mode.

KernelTime: Amount of time the process has spent running in kernel mode.

ImageName: String containing the process's image name.

BasePriority: Base priority of the process, which is the starting priority for threads created within the associated process.

UniqueProcessId: Process's unique process ID.

InheritedFromUniqueProcessId: Parent Process's unique process ID.

HandleCount: Current number of handles open in the process.

SessionId: Session identifier of the session the process is in.

SpareUl3: Reserved.

PeakVirtualSize: Peak size of virtual memory used by the process.

VirtualSize: Current size of virtual memory used by the process.

PageFaultCount: Number of page faults in the process.

PeakWorkingSetSize: Peak size of the working set of the process.

WorkingSetSize: Current size of the working set of the process.

QuotaPeakPagedPoolUsage: Peak quota charged to the process for paged pool usage.

QuotaPagedPoolUsage: Current quota charged to the process for paged pool usage.

QuotaPeakNonPagedPoolUsage: Peak quota charged to the process for nonpaged pool usage.

QuotaNonPagedPoolUsage: Current quota charged to the process for nonpaged pool usage.

PagefileUsage: Amount of bytes of page file storage in use by the process.

PeakPagefileUsage: Peak amount of bytes of page file storage in use by the process.

PrivatePageCount: Current number of memory pages allocated by the process.

2.2.2.10.1.1 NT6_TS_UNICODE_STRING

A UNICODE string.

```
typedef struct _NT6_TS_UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  [size_is(MaximumLength / 2), length_is(Length / 2)]
    PWSTR Buffer;
} NT6_TS_UNICODE_STRING;
```

Length: The actual length of the string currently stored in **Buffer**, in bytes.

MaximumLength: The maximum length of the string that could be stored in **Buffer**, in bytes.

Buffer: A wide character string. May not be NULL-terminated.

2.2.2.11 TS_COUNTER

A Terminal Services (TS) performance counter structure used to represent a single performance counter.

```
typedef struct _TS_COUNTER {
  TS_COUNTER_HEADER counterHead;
  DWORD dwValue;
  LARGE_INTEGER startTime;
} TS_COUNTER,
 *PTS_COUNTER;
```

counterHead: A header identifying the counter.

dwValue: The value of the counter. This indicates different things based on the counter.

startTime: Always set to zero currently because time stamps are not supported.

2.2.2.11.1 TS_COUNTER_HEADER

The header of the Terminal Services (TS) performance counter structure providing general information on the counter.

```
typedef struct _TS_COUNTER_HEADER {
  DWORD dwCounterID;
  BOOLEAN bResult;
} TS_COUNTER_HEADER,
 *PTS_COUNTER_HEADER;
```

dwCounterID: The identifier of the counter. Set by the caller of **RpcWinStationGetTermSrvCountersValue** to indicate the counter to retrieve data on. This

will be set to zero by **RpcWinStationGetTermSrvCountersValue** if the **dwCounterId** isn't recognized.

bResult: Set to TRUE if counter information is returned. Set to FALSE if counter data isn't being returned because the counter ID being requested was unrecognized.

The following counters are supported.

Total number of sessions: **dwCounterId** should be TERMSRV_TOTAL_SESSIONS. Value will indicate the total number of reconnections to the server since boot up.

Number of disconnected sessions: **dwCounterId** should be TERMSRV_DISC_SESSIONS. Value will indicate the total number of disconnections from the server since boot up.

Number of reconnected sessions: **dwCounterId** should be TERMSRV_RECON_SESSIONS. Value will indicate the total number of all reconnected sessions that have existed on the server since boot up.

Current number of active sessions: **dwCounterId** should be TERMSRV_CURRENT_ACTIVE_SESSIONS. Value will indicate the current number of active sessions on the server.

Current number of disconnected sessions: **dwCounterId** should be TERMSRV_CURRENT_DISC_SESSIONS. Value will indicate the current number of disconnected sessions on the server.<6>

2.2.2.12 USERCONFIG

For a specific Terminal Server session, indicates the user and session configuration.

```
#ifdef UNICODE
#define USERCONFIG USERCONFIGW
#define PUSERCONFIG PUSERCONFIGW
#else
#define USERCONFIG USERCONFIGA
#define PUSERCONFIG PUSERCONFIGA
#endif
```

2.2.2.12.1 USERCONFIGW

The **USERCONFIGW** structure indicates the user and session configuration for a specific Terminal Server session.

```
typedef struct _USERCONFIGW {
  ULONG fInheritAutoLogon  :1;
  ULONG fInheritResetBroken  :1;
  ULONG fInheritReconnectSame  :1;
  ULONG fInheritInitialProgram  :1;
  ULONG fInheritCallback  :1;
  ULONG fInheritCallbackNumber  :1;
  ULONG fInheritShadow  :1;
  ULONG fInheritMaxSessionTime  :1;
  ULONG fInheritMaxDisconnectionTime  :1;
  ULONG fInheritMaxIdleTime  :1;
  ULONG fInheritAutoClient  :1;
  ULONG fInheritSecurity  :1;
```

```
  ULONG fPromptForPassword  :1;
  ULONG fResetBroken  :1;
  ULONG fReconnectSame  :1;
  ULONG fLogonDisabled  :1;
  ULONG fWallPaperDisabled  :1;
  ULONG fAutoClientDrives  :1;
  ULONG fAutoClientLpts  :1;
  ULONG fForceClientLptDef  :1;
  ULONG fRequireEncryption  :1;
  ULONG fDisableEncryption  :1;
  ULONG fUnused1  :1;
  ULONG fHomeDirectoryMapRoot  :1;
  ULONG fUseDefaultGina  :1;
  ULONG fCursorBlinkDisabled  :1;
  ULONG fPublishedApp  :1;
  ULONG fHideTitleBar  :1;
  ULONG fMaximize  :1;
  ULONG fDisableCpm  :1;
  ULONG fDisableCdm  :1;
  ULONG fDisableCcm  :1;
  ULONG fDisableLPT  :1;
  ULONG fDisableClip  :1;
  ULONG fDisableExe  :1;
  ULONG fDisableCam  :1;
  ULONG fDisableAutoReconnect  :1;
  ULONG ColorDepth  :3;
  ULONG fInheritColorDepth  :1;
  ULONG fErrorInvalidProfile  :1;
  ULONG fPasswordIsScPin  :1;
  ULONG fDisablePNPRedir  :1;
  WCHAR UserName[USERNAME_LENGTH + 1];
  WCHAR Domain[DOMAIN_LENGTH + 1];
  WCHAR Password[PASSWORD_LENGTH + 1];
  WCHAR WorkDirectory[DIRECTORY_LENGTH + 1];
  WCHAR InitialProgram[INITIALPROGRAM_LENGTH + 1];
  WCHAR CallbackNumber[CALLBACK_LENGTH + 1];
  CALLBACKCLASS Callback;
  SHADOWCLASS Shadow;
  ULONG MaxConnectionTime;
  ULONG MaxDisconnectionTime;
  ULONG MaxIdleTime;
  ULONG KeyboardLayout;
  BYTE MinEncryptionLevel;
  WCHAR NWLogonServer[NASIFILESERVER_LENGTH + 1];
  APPLICATIONNAMEW PublishedName;
  WCHAR WFProfilePath[DIRECTORY_LENGTH + 1];
  WCHAR WFHomeDir[DIRECTORY_LENGTH + 1];
  WCHAR WFHomeDirDrive[4];
} USERCONFIGW,
 *PUSERCONFIGW;
```

fInheritAutoLogon: Prompt for the password setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritResetBroken: Reset the session when the connection is broken. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritReconnectSame: Reconnect from the same client setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritInitialProgram: The initial program setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritCallback: The callback setting. TRUE if machine/user policy is enforcing, or FALSE otherwise. Not used by RDP.

fInheritCallbackNumber: The callback number setting. TRUE if machine/user policy is enforcing, or FALSE otherwise. Not used by RDP.

fInheritShadow: The shadow setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritMaxSessionTime: The maximum allowed session connection time setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritMaxDisconnectionTime: The maximum allowed session disconnect time setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritMaxIdleTime: The maximum allowed session idle time. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritAutoClient: The auto client setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritSecurity: Inherit security setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fPromptForPassword: Set to TRUE to ignore credential send from client and always prompt for password, or FALSE otherwise.

fResetBroken: Set to TRUE to logoff the session when the idle timers for the session expire. Otherwise, the session will be disconnected when the timer expires.

fReconnectSame: Set to FALSE to indicate that the user can reconnect from any client computer to a disconnected session.

Set to TRUE to indicate that the user MUST reconnect to a disconnected session from the same client computer that initially established the disconnected session. Logging on from a different client computer will lead to a new Terminal Server session being created. This is only used by RDP.

fLogonDisabled: User cannot log into a session remotely. <7>

fWallPaperDisabled: Display of the desktop wallpaper in the session has been disabled.

fAutoClientDrives: Automatically redirect local drives on the client so they are accessible to the user in the remote Terminal Serversession.

fAutoClientLpts: Automatically redirect printers on the client so they are accessible to the user in the remote Terminal Serversession.

fForceClientLptDef: Force the client's redirected printer to be the default printer for the user.

fRequireEncryption: The connection must be encrypted.

fDisableEncryption: The connection does not need encryption.

fUnused1: Not used.

fHomeDirectoryMapRoot: Not used.

fUseDefaultGina: Override a third-party **GINA** so that only the default GINA is used for the Terminal Server session. <8>

fCursorBlinkDisabled: Disable the mouse cursor blinking. <9>

fPublishedApp: Not used.

fHideTitleBar: Not used.

fMaximize: Not used.

fDisableCpm: Disable client printer redirection.

fDisableCdm: Disable client drive redirection.

fDisableCcm: Disable client COM port redirection.

fDisableLPT: Disable client printer (LPT) port redirection.

fDisableClip: Disable client clipboard redirection.

fDisableExe: Disable .exe file execution.

fDisableCam: Disable client audio redirection.

fDisableAutoReconnect: Disable auto-reconnect functionality.<10>

ColorDepth: The color depth of the session. <11>

fInheritColorDepth: Set to TRUE to inherit color depth from the user or client configuration, or FALSE otherwise.<12>

fErrorInvalidProfile: Set to TRUE if WFProfilePath, WFHomeDir or WFHomeDirDrive is invalid (too long), or FALSE otherwise.<13>

fPasswordIsScPin: Set to TRUE if the password field contains a smartcard PIN. <14>

fDisablePNPRedir: Set to TRUE if Plug and Play (PnP) redirection is disabled, or FALSE otherwise.

UserName: The user name used in auto-logon scenarios. This field is inherited if **fInheritAutoLogon** is TRUE.

Domain: The domain name used in auto-logon scenarios. This field is inherited if **fInheritAutoLogon** is TRUE.

Password: The password used in auto-logon scenarios. This field is inherited if **fInheritAutoLogon** is TRUE.

WorkDirectory: The work directory for the initial program. This field is inherited if **fInheritInitialProgram** is TRUE.

InitialProgram: The program to run instead of explorer.exe if set. This field is inherited if **fInheritInitialProgram** is TRUE.

CallbackNumber: The number to call for callback operations. This field is inherited if **fInheritCallback** is TRUE. Not used by RDP.

Callback: The callback class for callback operations. This field is inherited if **fInheritCallback** is TRUE. Not used by RDP.

Shadow: The shadow setting of the session.

MaxConnectionTime: The maximum allowed session connection time setting of the session. The session will disconnect/logoff once the limit is reached.

MaxDisconnectionTime: The maximum allowed session disconnect time of the session. The session will logoff once the limit is reached.

MaxIdleTime: The maximum allowed session idle time setting of the session. The session will disconnect/logoff once the limit is reached.

KeyboardLayout: The keyboard layout (HKL) of the session.

MinEncryptionLevel: The minimum allowed encryption level of the session. This field is inherited if **fInheritSecurity** is TRUE. Not used by RDP.

NWLogonServer: The netware logon server name. Not used by Terminal Server RDP sessions.

PublishedName: Not used.

WFProfilePath: The Terminal Server profile path. Overrides the standard profile path.

WFHomeDir: Terminal Server home directory path. Overrides the standard home directory.

WFHomeDirDrive: Terminal Server home directory drive. Overrides the standard home directory.

2.2.2.12.2 USERCONFIGA

The **USERCONFIGA** structure indicates the user and session configuration for a specific Terminal Server session.

```
typedef struct _USERCONFIGA {
  ULONG fInheritAutoLogon  :1;
  ULONG fInheritResetBroken  :1;
  ULONG fInheritReconnectSame  :1;
  ULONG fInheritInitialProgram  :1;
  ULONG fInheritCallback  :1;
  ULONG fInheritCallbackNumber  :1;
  ULONG fInheritShadow  :1;
  ULONG fInheritMaxSessionTime  :1;
  ULONG fInheritMaxDisconnectionTime  :1;
  ULONG fInheritMaxIdleTime  :1;
  ULONG fInheritAutoClient  :1;
  ULONG fInheritSecurity  :1;
  ULONG fPromptForPassword  :1;
  ULONG fResetBroken  :1;
  ULONG fReconnectSame  :1;
  ULONG fLogonDisabled  :1;
```

```
  ULONG fWallPaperDisabled  :1;
  ULONG fAutoClientDrives  :1;
  ULONG fAutoClientLpts  :1;
  ULONG fForceClientLptDef  :1;
  ULONG fRequireEncryption  :1;
  ULONG fDisableEncryption  :1;
  ULONG fUnused1  :1;
  ULONG fHomeDirectoryMapRoot  :1;
  ULONG fUseDefaultGina  :1;
  ULONG fCursorBlinkDisabled  :1;
  ULONG fPublishedApp  :1;
  ULONG fHideTitleBar  :1;
  ULONG fMaximize  :1;
  ULONG fDisableCpm  :1;
  ULONG fDisableCdm  :1;
  ULONG fDisableCcm  :1;
  ULONG fDisableLPT  :1;
  ULONG fDisableClip  :1;
  ULONG fDisableExe  :1;
  ULONG fDisableCam  :1;
  ULONG fDisableAutoReconnect  :1;
  ULONG fErrorInvalidProfile  :1;
  ULONG fPasswordIsScPin  :1;
  ULONG fDisablePNPRedir  :1;
  CHAR UserName[USERNAME_LENGTH + 1];
  CHAR Domain[DOMAIN_LENGTH + 1];
  CHAR Password[PASSWORD_LENGTH + 1];
  CHAR WorkDirectory[DIRECTORY_LENGTH + 1];
  CHAR InitialProgram[INITIALPROGRAM_LENGTH + 1];
  CHAR CallbackNumber[CALLBACK_LENGTH + 1];
  CALLBACKCLASS Callback;
  SHADOWCLASS Shadow;
  ULONG MaxConnectionTime;
  ULONG MaxDisconnectionTime;
  ULONG MaxIdleTime;
  ULONG KeyboardLayout;
  BYTE MinEncryptionLevel;
  CHAR NWLogonServer[NASIFILESERVER_LENGTH + 1];
  APPLICATIONNAMEA PublishedName;
  CHAR WFProfilePath[DIRECTORY_LENGTH + 1];
  CHAR WFHomeDir[DIRECTORY_LENGTH + 1];
  CHAR WFHomeDirDrive[4];
} USERCONFIGA,
 *PUSERCONFIGA;
```

fInheritAutoLogon: Prompt for password setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritResetBroken: Reset the session when the connection is broken. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritReconnectSame: Reconnect from the same client setting. TRUE if machine/user policy is enforcing, or FALSE otherwise

fInheritInitialProgram: The initial program setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritCallback: The callback setting. TRUE if machine/user policy is enforcing, or FALSE otherwise. Not used by RDP.

fInheritCallbackNumber: The callback number setting. TRUE if machine/user policy is enforcing, or FALSE otherwise. Not used by RDP.

fInheritShadow: The shadow setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritMaxSessionTime: The maximum allowed session connection time setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritMaxDisconnectionTime: The maximum allowed session disconnect time setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritMaxIdleTime: The maximum allowed session idle time. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritAutoClient: The auto client setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fInheritSecurity: The inherit security setting. TRUE if machine/user policy is enforcing, or FALSE otherwise.

fPromptForPassword: Set to TRUE to ignore the credential sent from the client and always prompt for password, or FALSE otherwise.

fResetBroken: Set to TRUE to logoff the session when the idle timers for the session expire. Otherwise, the session will be disconnected when the timer expires.

fReconnectSame: Set to FALSE to indicate that the user can reconnect from any client computer to a disconnected session.

Set to TRUE to indicate that the user MUST reconnect to a disconnected session from the same client computer that initially established the disconnected session. Logging on from a different client computer will lead to a new Terminal Server session being created. This is only used by RDP.

fLogonDisabled: Set to TRUE to specify that the user cannot log into a session remotely.<15>

fWallPaperDisabled: Set to TRUE to specify that display of the desktop wallpaper in the session has been disabled.

fAutoClientDrives: Automatically redirect local drives on the client so they are accessible to the user in the remote Terminal Server session.

fAutoClientLpts: Automatically redirect printers on the client so they are accessible to the user in the remote Terminal Server session.

fForceClientLptDef: Force the client's redirected printer to be the default printer for the user.

fRequireEncryption: The connection must be encrypted.

fDisableEncryption: The connection does not need encryption.

fUnused1: Not used.

fHomeDirectoryMapRoot: Not used.

fUseDefaultGina: Override a third-party GINA so that only the default GINA is used for the Terminal Server session.<16>

fCursorBlinkDisabled: Disable the mouse cursor blinking.<17>

fPublishedApp: Not used.

fHideTitleBar: Not used.

fMaximize: Not used.

fDisableCpm: Disable client printer redirection.

fDisableCdm: Disable client drive redirection.

fDisableCcm: Disable client COM port redirection.

fDisableLPT: Disable client printer (LPT) port redirection.

fDisableClip: Disable client clipboard redirection.

fDisableExe: Disable .exe file execution.

fDisableCam: Disable client audio-redirection.

fDisableAutoReconnect: Disable auto-reconnect functionality.<18>

fErrorInvalidProfile: Set to TRUE if WFProfilePath, WFHomeDir or WFHomeDirDrive is invalid (too long).<19>

fPasswordIsScPin: Set to TRUE if the password field contains a smartcard PIN.<20>

fDisablePNPRedir: Set to TRUE if Plug and Play (PnP) redirection is disabled, or FALSE otherwise.

UserName: The user name used in auto-logon scenarios. This field is inherited if **fInheritAutoLogon** is TRUE.

Domain: The domain name used in auto-logon scenarios. This field is inherited if **fInheritAutoLogon** is TRUE.

Password: The password used in auto-logon scenarios. This field is inherited if **fInheritAutoLogon** is TRUE.

WorkDirectory: The work directory for the initial program. This field is inherited if **fInheritInitialProgram** is TRUE.

InitialProgram: The program to run instead of explorer.exe if set. This field is inherited if **fInheritInitialProgram** is TRUE.

CallbackNumber: The number to call for callback operations. This field is inherited if **fInheritCallback** is TRUE. Not used by RDP.

Callback: The callback class for callback operations. This field is inherited if **fInheritCallback** is TRUE. Not used by RDP.

Shadow: The shadow setting of the session.

MaxConnectionTime: The maximum allowed session connection time setting of the session. The session will disconnect/logoff once the limit is reached.

MaxDisconnectionTime: The maximum allowed session disconnect time of the session. The session will logoff once the limit is reached.

MaxIdleTime: The maximum allowed session idle time setting of the session. The session will disconnect/logoff once the limit is reached.

KeyboardLayout: The keyboard layout (HKL) of the session.

MinEncryptionLevel: The minimum allowed encryption level of the session. This field is inherited if **fInheritSecurity** is TRUE. Not used by RDP.

NWLogonServer: The netware logon server name. Not used by Terminal Server RDP sessions.

PublishedName: Not used.

WFProfilePath: Terminal Server profile path. Overrides the standard profile path.

WFHomeDir: Terminal Server home directory path. Overrides the standard home directory.

WFHomeDirDrive: Terminal Server home directory drive. Overrides the standard home directory.

2.2.2.12.3 CALLBACKCLASS

The class used for callback options to indicate the type of callback. This is not used by RDP.

```
typedef  enum _CALLBACKCLASS
{
  Callback_Disable,
  Callback_Roving,
  Callback_Fixed
} CALLBACKCLASS;
```

Callback_Disable: Callback is disabled.

Callback_Roving: The callback number is a roving number.

Callback_Fixed: The callback number is a fixed number.

2.2.2.12.4 APPLICATIONNAME

The data type representing an application name.

```
typedef WCHAR APPLICATIONNAMEW[MAX_BR_NAME];

typedef WCHAR* PAPPLICATIONNAMEW;

typedef CHAR APPLICATIONNAMEA[MAX_BR_NAME];

typedef CHAR* PAPPLICATIONNAMEA;
```

The following block determines **APPLICATIONNAME**:

```
#ifdef UNICODE
#define APPLICATIONNAME APPLICATIONNAMEW
#define PAPPLICATIONNAME PAPPLICATIONNAMEW
#else
#define APPLICATIONNAME APPLICATIONNAMEA
#define PAPPLICATIONNAME PAPPLICATIONNAMEA
#endif
```

2.2.2.13 WINSTATIONCLIENT

The WINSTATIONCLIENT structure defines the client-requested configuration when connecting to a session.

```
#ifdef UNICODE
#define WINSTATIONCLIENT WINSTATIONCLIENTW
#define PWINSTATIONCLIENT PWINSTATIONCLIENTW
#else
#define WINSTATIONCLIENT WINSTATIONCLIENTA
#define PWINSTATIONCLIENT PWINSTATIONCLIENTA
#endif
```

2.2.2.13.1 WINSTATIONCLIENTW

The **WINSTATIONCLIENTW** structure defines the client-requested configuration when connecting to a session.

```
typedef struct _WINSTATIONCLIENTW {
  ULONG fTextOnly  :1;
  ULONG fDisableCtrlAltDel  :1;
  ULONG fMouse  :1;
  ULONG fDoubleClickDetect  :1;
  ULONG fINetClient  :1;
  ULONG fPromptForPassword  :1;
  ULONG fMaximizeShell  :1;
  ULONG fEnableWindowsKey  :1;
  ULONG fRemoteConsoleAudio  :1;
  ULONG fPasswordIsScPin  :1;
  ULONG fNoAudioPlayback  :1;
  ULONG fUsingSavedCreds  :1;
  WCHAR ClientName[CLIENTNAME_LENGTH + 1 ];
  WCHAR Domain[DOMAIN_LENGTH + 1 ];
  WCHAR UserName[USERNAME_LENGTH + 1 ];
  WCHAR Password[PASSWORD_LENGTH + 1 ];
  WCHAR WorkDirectory[DIRECTORY_LENGTH + 1 ];
  WCHAR InitialProgram[INITIALPROGRAM_LENGTH + 1 ];
  ULONG SerialNumber;
  BYTE EncryptionLevel;
  ULONG ClientAddressFamily;
  WCHAR ClientAddress[CLIENTADDRESS_LENGTH + 1 ];
```

```
  USHORT HRes;
  USHORT VRes;
  USHORT ColorDepth;
  USHORT ProtocolType;
  ULONG KeyboardLayout;
  ULONG KeyboardType;
  ULONG KeyboardSubType;
  ULONG KeyboardFunctionKey;
  WCHAR imeFileName[IMEFILENAME_LENGTH + 1 ];
  WCHAR ClientDirectory[DIRECTORY_LENGTH + 1 ];
  WCHAR ClientLicense[CLIENTLICENSE_LENGTH + 1 ];
  WCHAR ClientModem[CLIENTMODEM_LENGTH + 1 ];
  ULONG ClientBuildNumber;
  ULONG ClientHardwareId;
  USHORT ClientProductId;
  USHORT OutBufCountHost;
  USHORT OutBufCountClient;
  USHORT OutBufLength;
  WCHAR AudioDriverName[9];
  TS_TIME_ZONE_INFORMATION ClientTimeZone;
  ULONG ClientSessionId;
  WCHAR clientDigProductId[CLIENT_PRODUCT_ID_LENGTH];
  ULONG PerformanceFlags;
  ULONG ActiveInputLocale;
} WINSTATIONCLIENTW,
 *PWINSTATIONCLIENTW;
```

fTextOnly: Text-only client session. This is always FALSE.

fDisableCtrlAltDel: Set to TRUE to specify that CTRL+ALT+DEL is disabled.

fMouse: Mouse connected to client. Usually TRUE.

fDoubleClickDetect: Double-click detect flag. TRUE indicates detect double-click, FALSE otherwise.

fINetClient: Not used by Terminal Server RDP sessions. Always set to FALSE.

fPromptForPassword: If TRUE, user will always be prompted for password, even if password is saved from previous connection.

fMaximizeShell: TRUE indicates maximize the shell, FALSE otherwise.

fEnableWindowsKey: Windows key enabled in Terminal Server session.

fRemoteConsoleAudio: Set to TRUE if audio for the console session is left remotely at the server, or FALSE otherwise. <21>

fPasswordIsScPin: Set to TRUE if the password field contains a smartcard PIN, or FALSE otherwise.<22>

fNoAudioPlayback: Set to TRUE if audio shouldn't be played back, or FALSE otherwise. <23>

fUsingSavedCreds: Set to TRUE if the Terminal Server connection was made using a credential saved on the client machine, or FALSE otherwise.

ClientName: The name of the client machine.

Domain: The user's domain name.

UserName: The user's user name.

Password: The user's password.

WorkDirectory: The work directory for the initial program.

InitialProgram: The program to run instead of explorer.exe if set.

SerialNumber: The client computer's unique serial number.

EncryptionLevel: The encryption level.

ClientAddressFamily: The address family of the client's address. For Terminal Server RDP connections, this will be AF_INET for IPv4 or AF_INET6 for IPv6. <24>

ClientAddress: The client's address. The format depends on the value of **ClientAddressFamily**. See [MSDN-SOCKET] for more information.

HRes: The horizontal resolution, in pixels.

VRes: The vertical resolution, in pixels.

ColorDepth: The color depth. The supported values 1, 2, 4, 8, 16 are translated, respectively, as the following number of colors supported: 16 (4 bpp), 256 (8 bpp), 65,536 (16 bpp), 16 million (24 bpp), 32,768 (15 bpp).<25>

ProtocolType: The type of protocol. PROTOCOL_OTHERS, PROTOCOL_RDP or PROTOCOL_CONSOLE. <26>

KeyboardLayout: The keyboard layout.

KeyboardType: The keyboard type.

KeyboardSubType: The keyboard subtype.

KeyboardFunctionKey: The keyboard function type.

imeFileName: The file name of the Input Method Editor (IME), if any, used for the session. For more information on Input Method Editors, see [MSFT-IME].

ClientDirectory: The directory where the client was installed.

ClientLicense: The client's license. Not used for Terminal Server RDP sessions.

ClientModem: The client's modem. Not used for Terminal Server RDP sessions.

ClientBuildNumber: The client's build number.

ClientHardwareId: The client-specific hardware identifier.

ClientProductId: The client-specific product identifier.

OutBufCountHost: The number of output buffers on the host machine.

OutBufCountClient: The number of output buffers on the client machine.

OutBufLength: The length of the output buffer, in bytes.

AudioDriverName: The audio driver's name.

ClientTimeZone: The client's time zone. <27>

ClientSessionId: The client's session ID.<28>

clientDigProductId: The client-specific product ID.<29>.

PerformanceFlags: RDP-specific performance flags.<30>Can be any combination of the following except for **TS_PERF_DISABLE_NOTHING**.

Value	Meaning
TS_PERF_DISABLE_NOTHING 0x00000000	Disable nothing.
TS_PERF_DISABLE_WALLPAPER 0x00000001	Disable wallpaper.
TS_PERF_DISABLE_FULLWINDOWDRAG 0x00000002	Disable full window drag animation.
TS_PERF_DISABLE_MENUANIMATIONS 0x00000004	Disable menu animations.
TS_PERF_DISABLE_THEMING 0x00000008	Disable themes.
TS_PERF_ENABLE_ENHANCED_GRAPHICS 0x00000010	Enable enhanced graphics.
TS_PERF_DISABLE_CURSOR_SHADOW 0x00000020	Disable cursor shadow.
TS_PERF_DISABLE_CURSORSETTINGS 0x00000040	Disable cursor settings.
TS_PERF_ENABLE_FONT_SMOOTHING 0x00000080	Enable font smoothing.<31>
TS_PERF_ENABLE_DESKTOP_COMPOSITION 0x00000100	Enable desktop composition.<32>
TS_PERF_DEFAULT_NONPERFCLIENT_SETTING 0x40000000	Set internally for clients not aware of this setting.<33>
TS_PERF_RESERVED1 0x80000000	Reserved and used internally by the client.

ActiveInputLocale: Client language locale HKL.<34>

2.2.2.13.2 WINSTATIONCLIENTA

The **WINSTATIONCLIENTA** structure defines the client-requested configuration when connecting to a session.

```
typedef struct _WINSTATIONCLIENTA {
  ULONG fTextOnly  :1;
  ULONG fDisableCtrlAltDel  :1;
  ULONG fMouse  :1;
  ULONG fDoubleClickDetect  :1;
  ULONG fINetClient  :1;
  ULONG fPromptForPassword  :1;
  ULONG fMaximizeShell  :1;
  ULONG fEnableWindowsKey  :1;
  ULONG fRemoteConsoleAudio  :1;
  ULONG fPasswordIsScPin  :1;
  ULONG fNoAudioPlayback  :1;
  ULONG fUsingSavedCreds  :1;
  char ClientName[CLIENTNAME_LENGTH + 1];
  char Domain[DOMAIN_LENGTH + 1];
  char UserName[USERNAME_LENGTH + 1];
  char Password[PASSWORD_LENGTH + 1];
  char WorkDirectory[DIRECTORY_LENGTH + 1];
  char InitialProgram[INITIALPROGRAM_LENGTH + 1];
  ULONG SerialNumber;
  BYTE EncryptionLevel;
  ULONG ClientAddressFamily;
  char ClientAddress[CLIENTADDRESS_LENGTH + 1];
  USHORT HRes;
  USHORT VRes;
  USHORT ColorDepth;
  USHORT ProtocolType;
  ULONG KeyboardLayout;
  ULONG KeyboardType;
  ULONG KeyboardSubType;
  ULONG KeyboardFunctionKey;
  char imeFileName[IMEFILENAME_LENGTH + 1];
  char ClientDirectory[DIRECTORY_LENGTH + 1];
  char ClientLicense[CLIENTLICENSE_LENGTH + 1];
  char ClientModem[CLIENTMODEM_LENGTH + 1];
  ULONG ClientBuildNumber;
  ULONG ClientHardwareId;
  USHORT ClientProductId;
  USHORT OutBufCountHost;
  USHORT OutBufCountClient;
  USHORT OutBufLength;
  char AudioDriverName[9];
  TS_TIME_ZONE_INFORMATION ClientTimeZone;
  ULONG ClientSessionId;
  char clientDigProductId[CLIENT_PRODUCT_ID_LENGTH];
  ULONG PerformanceFlags;
  ULONG ActiveInputLocale;
} WINSTATIONCLIENTA,
 *PWINSTATIONCLIENTA;
```

fTextOnly: Text-only client session. This is always FALSE.

fDisableCtrlAltDel: Set to TRUE to specify that Ctrl-Alt-Del is disabled.

fMouse: Mouse connected to client. Usually TRUE.

fDoubleClickDetect: Double-click detect flag. TRUE indicates detect double-click, FALSE otherwise.

fINetClient: Not used by Terminal Server RDP sessions. Always set to FALSE.

fPromptForPassword: If TRUE, user will always be prompted for password, even if password is saved from previous connection.

fMaximizeShell: TRUE indicates maximize the shell, false otherwise.

fEnableWindowsKey: Windows key enabled in TS session.

fRemoteConsoleAudio: Set to TRUE if audio for the console session is left remotely at the server, or FALSE otherwise. <35>

fPasswordIsScPin: Set to TRUE if the password field contains a smartcard PIN, or FALSE otherwise. <36>

fNoAudioPlayback: Set to TRUE if audio shouldn't be played back, or FALSE otherwise. <37>

fUsingSavedCreds: Set to TRUE if the Terminal Server connection was made using a credential saved on the client machine; FALSE otherwise.

ClientName: The name of the client machine.

Domain: The user's domain name.

UserName: The user's user name.

Password: The user's password.

WorkDirectory: The work directory for the initial program.

InitialProgram: The program to run instead of explorer.exe if set.

SerialNumber: The client computer's unique serial number.

EncryptionLevel: The encryption level.

ClientAddressFamily: The address family of the client's address. For Terminal ServerRDP connections, this will be AF_INET for IPv4 or AF_INET6 for IPv6. <38>

ClientAddress: The client's address. The format depends on the value of ClientAddressFamily. See [MSDN-SOCKET] for more information.

HRes: The horizontal resolution, in pixels.

VRes: The vertical resolution, in pixels.

ColorDepth: The color depth. The supported values 1, 2, 4, 8, 16 are translated, respectively, as the following number of colors supported: 16 (4 bpp), 256 (8 bpp), 65,536 (16 bpp), 16 million (24 bpp), 32,768 (15 bpp). <39>

ProtocolType: The type of protocol. PROTOCOL_OTHERS, PROTOCOL_RDP or PROTOCOL_CONSOLE. <40>

KeyboardLayout: The keyboard layout.

KeyboardType: The keyboard type.

KeyboardSubType: The keyboard subtype.

KeyboardFunctionKey: The keyboard function type.

imeFileName: The file name of the Input Method Editor (IME), if any, used for the session. For more information on Input Method Editors, see [MSFT-IME].

ClientDirectory: The directory where the client was installed.

ClientLicense: The client's license. Not used for Terminal Server RDP sessions.

ClientModem: The client's modem. Not used for Terminal Server RDP sessions.

ClientBuildNumber: The client's build number.

ClientHardwareId: The client-specific hardware identifier.

ClientProductId: The client-specific product identifier.

OutBufCountHost: The number of output buffers on the host machine.

OutBufCountClient: The number of output buffers on the client machine.

OutBufLength: The length of the output buffer, in bytes.

AudioDriverName: The audio driver's name.

ClientTimeZone: The client's time zone. <41>

ClientSessionId: The client's session ID. <42>

clientDigProductId: The client-specific product ID. <43>

PerformanceFlags: RDP-specific performance flags. <44> Can be any combination of the following except for TS_PERF_DISABLE_NOTHING.

Value	Meaning
TS_PERF_DISABLE_NOTHING 0x00000000	Disable nothing.
TS_PERF_DISABLE_WALLPAPER 0x00000001	Disable wallpaper.
TS_PERF_DISABLE_FULLWINDOWDRAG 0x00000002	Disable full window drag animation.
TS_PERF_DISABLE_MENUANIMATIONS 0x00000004	Disable menu animations.
TS_PERF_DISABLE_THEMING 0x00000008	Disable themes.
TS_PERF_ENABLE_ENHANCED_GRAPHICS 0x00000010	Enable enhanced graphics.
TS_PERF_DISABLE_CURSOR_SHADOW 0x00000020	Disable cursor shadow.

Value	Meaning
TS_PERF_DISABLE_CURSORSETTINGS 0x00000040	Disable cursor settings.
TS_PERF_ENABLE_FONT_SMOOTHING 0x00000080	Enable font smoothing. <45>
TS_PERF_ENABLE_DESKTOP_COMPOSITION 0x00000100	Enable desktop composition. <46>
TS_PERF_DEFAULT_NONPERFCLIENT_SETTING 0x40000000	Set internally for clients not aware of this setting.<47>
TS_PERF_RESERVED1 0x80000000	Reserved and used internally by the client.

ActiveInputLocale: Client language locale HKL. <48>

2.2.2.13.3 TS_TIME_ZONE_INFORMATION

The **TS_TIME_ZONE_INFORMATION** structure contains client time zone information.

```
typedef struct _TS_TIME_ZONE_INFORMATION {
  LONG Bias;
  WCHAR StandardName[32 ];
  TS_SYSTEMTIME StandardDate;
  LONG StandardBias;
  WCHAR DaylightName[32 ];
  TS_SYSTEMTIME DaylightDate;
  LONG DaylightBias;
} TS_TIME_ZONE_INFORMATION;
```

Bias: A 32-bit unsigned integer. Current bias for local time translation on the client, in minutes. The bias is the difference, in minutes, between Coordinated Universal Time (UTC) and local time. All translations between UTC and local time are based on the following formula:

UTC = local time + bias

StandardName: A description for standard time on the client. For example, this field could contain the string "Pacific Standard Time" to indicate Pacific Standard Time. An array of 32 Unicode characters.

StandardDate: A TS_SYSTEMTIME structure that contains the date and local time when the transition from daylight saving time to standard time occurs on the client. If this field is specified, the **DaylightDate** field is also specified.

StandardBias: A 32-bit unsigned integer. Bias value to be used during local time translations that occur during standard time. This field should be ignored if a value is not supplied in the **StandardDate** field. This value is added to the value of the **Bias** field to form the bias used during standard time. In most time zones, the value of this field is 0.

DaylightName: A description for daylight time on the client. For example, this field could contain "Pacific Daylight Time" to indicate Pacific Daylight Time. An array of 32 Unicode characters.

DaylightDate: A TS_SYSTEMTIME that contains a date and local time when the transition from standard time to daylight saving time occurs on the client. If this field is specified, the **StandardDate** field is also specified.

DaylightBias: A 32-bit unsigned integer. Bias value to be used during local time translations that occur during daylight saving time. This field should be ignored if a value for the **DaylightDate** field is not supplied. This value is added to the value of the **Bias** field to form the bias used during daylight saving time. In most time zones, the value of this field is 60.

2.2.2.13.3.1 TS_SYSTEMTIME

Information about a time zone. This structure is identical to the structure **SYSTEMTIME**. For more information, see [MSDN-SYSTIME].

```
typedef struct _TS_SYSTEMTIME {
  USHORT wYear;
  USHORT wMonth;
  USHORT wDayOfWeek;
  USHORT wDay;
  USHORT wHour;
  USHORT wMinute;
  USHORT wSecond;
  USHORT wMilliseconds;
} TS_SYSTEMTIME;
```

wYear: The year when transition occurs (1601 to 30827).

wMonth: The month when transition occurs.

Value	Meaning
1	January
2	February
3	March
4	April
5	May
6	June
7	July
8	August
9	September
10	October
11	November
12	December

wDayOfWeek: The day of the week when transition occurs.

Value	Meaning
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

wDay: The occurrence of **wDayOfWeek** within the month when the transition takes place.

Value	Meaning
1	First occurrence of **wDayOfWeek**
2	Second occurrence of **wDayOfWeek**
3	Third occurrence of **wDayOfWeek**
4	Fourth occurrence of **wDayOfWeek**
5	Last occurrence of **wDayOfWeek**

wHour: The hour when transition occurs (0 to 23).

wMinute: The minute when transition occurs (0 to 59).

wSecond: The second when transition occurs (0 to 59).

wMilliseconds: The millisecond when transition occurs (0 to 999).

2.2.2.14 WINSTATIONINFORMATION

Provides the current values of various properties such as state, connect time, last input time, and so on, for a session.

```
#ifdef UNICODE
#define WINSTATIONINFORMATION WINSTATIONINFORMATIONW
#define PWINSTATIONINFORMATION PWINSTATIONINFORMATIONW
#else
#define WINSTATIONINFORMATION WINSTATIONINFORMATIONA
#define PWINSTATIONINFORMATION PWINSTATIONINFORMATIONA
#endif
```

2.2.2.14.1 WINSTATIONINFORMATIONW

The **WINSTATIONINFORMATIONW** structure defines the Unicode character version of WINSTATIONINFORMATION.

```
typedef struct _WINSTATIONINFORMATIONW {
  WINSTATIONSTATECLASS ConnectState;
  WINSTATIONNAMEW WinStationName;
  ULONG LogonId;
  LARGE_INTEGER ConnectTime;
  LARGE_INTEGER DisconnectTime;
  LARGE_INTEGER LastInputTime;
  LARGE_INTEGER LogonTime;
  PROTOCOLSTATUS Status;
  WCHAR Domain[DOMAIN_LENGTH + 1];
  WCHAR UserName[USERNAME_LENGTH + 1];
  LARGE_INTEGER CurrentTime;
} WINSTATIONINFORMATIONW,
 *PWINSTATIONINFORMATIONW;
```

ConnectState: The current connect state of the session.

WinStationName: The name of the session.

LogonId: The session identifier of the session.

ConnectTime: The time the session was connected to.

DisconnectTime: The time the session was disconnected from.

LastInputTime: The time the session last received input. This is an indicator of how long a session has been idle.

LogonTime: The time the session was logged on to.

Status: The status of the protocol, as specified in section 2.2.2.14.3.

Domain: The user's domain name.

UserName: The user's user name.

CurrentTime: The current time in the session.

2.2.2.14.2 WINSTATIONINFORMATIONA

The **WINSTATIONINFORMATIONA** structure defines the ANSI character version of WINSTATIONINFORMATION.

```
typedef struct _WINSTATIONINFORMATIONW {
  WINSTATIONSTATECLASS ConnectState;
  WINSTATIONNAMEA WinStationName;
  ULONG LogonId;
  LARGE_INTEGER ConnectTime;
  LARGE_INTEGER DisconnectTime;
  LARGE_INTEGER LastInputTime;
  LARGE_INTEGER LogonTime;
  PROTOCOLSTATUS Status;
  CHAR Domain[DOMAIN_LENGTH + 1];
  CHAR UserName[USERNAME_LENGTH + 1];
  LARGE_INTEGER CurrentTime;
```

```
} WINSTATIONINFORMATIONA,
 *PWINSTATIONINFORMATIONA;
```

ConnectState: The current connect state of the session.

WinStationName: The name of the session.

LogonId: The session identifier of the session.

ConnectTime: The time the session was connected to.

DisconnectTime: The time the session was disconnected from.

LastInputTime: The time the session last received input. This is an indicator of how long a session has been idle.

LogonTime: The time the session was logged on to.

Status: The status of the protocol, as specified in section 2.2.2.14.3.

Domain: The user's domain name.

UserName: The user's user name.

CurrentTime: The current time in the session.

2.2.2.14.3 PROTOCOLSTATUS

The status of the protocol used by the session.

```
typedef struct _PROTOCOLSTATUS {
  PROTOCOLCOUNTERS Output;
  PROTOCOLCOUNTERS Input;
  CACHE_STATISTICS Cache;
  ULONG AsyncSignal;
  ULONG AsyncSignalMask;
} PROTOCOLSTATUS,
 *PPROTOCOLSTATUS;
```

Output: Output protocol counters.

Input: Input protocol counters.

Cache: Statistics of the cache.

AsyncSignal: Indicator of async signal, such as MS_CTS_ON, for async protocols. For more information on asynchronous protocols, see [MSDN-SERIAL].

AsyncSignalMask: Mask of async signal events, such as EV_CTS, for async protocols. For more information on asynchronous protocols, see [MSDN-SERIAL].

2.2.2.14.3.1 PROTOCOLSTATUSEX

The **PROTOCOLSTATUSEX** structure defines the extended status of the protocol used by the session.

```
typedef struct {
  PROTOCOLSTATUS ProtocolStatus;
  LARGE_INTEGER Counters[MAX_COUNTER_EXTENSIONS];
} PROTOCOLSTATUSEX,
 *PPROTOCOLSTATUSEX;
```

ProtocolStatus: The status of the protocol as described in section 2.2.2.14.3. In C++, this element is inherited; otherwise it is directly included.

Counters: The value of the various counters associated with the protocol as specified in **PROTOCOLCOUNTERS**.

2.2.2.14.3.2 PROTOCOLCOUNTERS

Protocol performance counters.

```
typedef struct _PROTOCOLCOUNTERS {
  ULONG WdBytes;
  ULONG WdFrames;
  ULONG WaitForOutBuf;
  ULONG Frames;
  ULONG Bytes;
  ULONG CompressedBytes;
  ULONG CompressFlushes;
  ULONG Errors;
  ULONG Timeouts;
  ULONG AsyncFramingError;
  ULONG AsyncOverrunError;
  ULONG AsyncOverflowError;
  ULONG AsyncParityError;
  ULONG TdErrors;
  USHORT ProtocolType;
  USHORT Length;
  union {
    TSHARE_COUNTERS TShareCounters;
    ULONG Reserved[100];
  } Specific;
} PROTOCOLCOUNTERS,
 *PPROTOCOLCOUNTERS;
```

WdBytes: Winstation driver number of bytes sent and received.

WdFrames: Winstation driver number of frames sent and received.

WaitForOutBuf: The number of times waited for an output buffer to become available.

Frames: Transport driver number of frames.

Bytes: Transport driver number of bytes.

CompressedBytes: Number of compressed bytes.

CompressFlushes: Number of compress flushes. A compress flush is when compression for a packet failed and the original uncompressed packet replaces it.

Errors: Number of packets that were in error during the session.

Timeouts: Number of timeouts.

AsyncFramingError: Number of async framing errors.

AsyncOverrunError: Number of async overrun errors.

AsyncOverflowError: Number of async overflow errors.

AsyncParityError: Number of async parity errors.

TdErrors: Number of transport protocol errors.

ProtocolType: Protocol type. Can be PROTOCOL_RDP for RDP protocol or PROTOCOL_OTHERS for other protocols.

Length: Length of data in protocol-specific area. Can be up to 100 * sizeof(ULONG) in size.

Specific: Specifies which types of counters are to be queried. It can be one of the following:

TShareCounters: RDP-specific protocol performance counters.

Reserved: Reserved for future use.

2.2.2.14.3.2.1 TSHARE_COUNTERS

RDP-specific protocol performance counters.<49> Not used.

```
typedef struct _TSHARE_COUNTERS {
  ULONG Reserved;
} TSHARE_COUNTERS,
 *PTSHARE_COUNTERS;
```

Reserved: Reserved value.

2.2.2.14.3.3 CACHE_STATISTICS

Cache statistics on the protocol.

```
typedef struct CACHE_STATISTICS {
  USHORT ProtocolType;
  USHORT Length;
  union {
    RESERVED_CACHE ReservedCacheStats;
    TSHARE_CACHE TShareCacheStats;
    ULONG Reserved[20];
  } Specific;
```

```
} CACHE_STATISTICS,
 *PCACHE_STATISTICS;
```

ProtocolType: Protocol type. Can be PROTOCOL_RDP for RDP protocol or PROTOCOL_OTHERS for other protocols.

Length: Length of data in protocol-specific area. Can be up to 20 * sizeof(ULONG) in size.

Specific: The union of the following members:

ReservedCacheStats: This structure is not used by Terminal Services.

TShareCacheStats: RDP-specific cache statistics.

Reserved: Reserved for future use.

2.2.2.14.3.3.1 RESERVED_CACHE

Cache statistics.<50>

```
typedef struct _RESERVED_CACHE {
  THINWIRECACHE ThinWireCache[MAX_THINWIRECACHE ];
} RESERVED_CACHE,
 *PRESERVED_CACHE;
```

ThinWireCache: The ThinWireCache structure used for Terminal Server's display for RESERVED_CACHE.

2.2.2.14.3.3.1.1 THINWIRECACHE

The ThinWireCache structure used for Terminal Server display for RESERVED_CACHE.

```
typedef struct _THINWIRECACHE {
  ULONG CacheReads;
  ULONG CacheHits;
} THINWIRECACHE,
 *PTHINWIRECACHE;
```

CacheReads: Number of cache reads.

CacheHits: Number of cache hits.

2.2.2.14.3.3.2 TSHARE_CACHE

Reserved. Not currently used.

```
typedef struct _TSHARE_CACHE {
  ULONG Reserved;
} TSHARE_CACHE,
```

```
*PTSHARE_CACHE;
```

Reserved: Reserved. Not currently used.

2.2.2.15 PDPARAMS

The protocol driver parameters structure. The core Terminal Services binaries only read this data from the system data store and pass it to callers of **RpcWinStationQueryInformation** and other places. There is no interpretation of the configuration data. The actual use of this configuration data is in lower-level protocol drivers.

```
#ifdef UNICODE
#define PDPARAMS PDPARAMSW
#define PPDPARAMS PPDPARAMSW
#else
#define PDPARAMS PDPARAMSA
#define PPDPARAMS PPDPARAMSA
#endif
```

2.2.2.15.1 PDPARAMSW

The **PDPARAMSW** structure defines the Unicode character version of PDPARAMS.

```
typedef struct _PDPARAMSW {
  SDCLASS SdClass;
  union {
    NETWORKCONFIGW Network;
    ASYNCCONFIGW Async;
    NASICONFIGW Nasi;
    OEMTDCONFIGW OemTd;
  };
} PDPARAMSW,
 *PPDPARAMSW;
```

SdClass: Stack driver class. Indicates which one of the union's structures is valid and should be looked at.

Network: Configuration of network drivers. Used if **SdClass** is SdNetwork.

Async: Configuration of async (modem) driver. Used if **SdClass** is SdAsync. Not used by Terminal Server RDP sessions.

Nasi: Reserved.

OemTd: Configuration of OEM transport driver. Used if **SdClass** is SdOemTransport. Not used by TS RDP sessions.

2.2.2.15.2 PDPARAMSA

The **PDPARAMSA** structure defines the ANSI character version of PDPARAMS.

```
typedef struct _PDPARAMSA {
  SDCLASS SdClass;
  union {
    NETWORKCONFIGA Network;
    ASYNCCONFIGA Async;
    NASICONFIGA Nasi;
    OEMTDCONFIGA OemTd;
  };
} PDPARAMSA,
 *PPDPARAMSA;
```

SdClass: Stack driver class. Indicates which one of the union's structures is valid and should be looked at.

Network: Configuration of network drivers. Used if **SdClass** is SdNetwork.

Async: Configuration of async (modem) driver. Used if **SdClass** is SdAsync. Not used by Terminal Server (TS) RDP sessions.

Nasi: Reserved.

OemTd: Configuration of OEM transport driver. Used if **SdClass** is SdOemTransport. Not used by TS RDP sessions.

2.2.2.16 NETWORKCONFIG

The network protocol driver's configuration structure. The following block determines NETWORKCONFIG.

```
#ifdef UNICODE
#define NETWORKCONFIG NETWORKCONFIGW
#define PNETWORKCONFIG PNETWORKCONFIGW
#else
#define NETWORKCONFIG NETWORKCONFIGA
#define PNETWORKCONFIG PNETWORKCONFIGA
#endif
```

2.2.2.16.1 NETWORKCONFIGW

The **NETWORKCONFIGW** structure defines the Unicode character version of NETWORKCONFIG.

```
typedef struct _NETWORKCONFIGW {
  LONG LanAdapter;
  DEVICENAMEW NetworkName;
  ULONG Flags;
} NETWORKCONFIGW,
 *PNETWORKCONFIGW;
```

LanAdapter: The LANA ID of the network adaptor.

NetworkName: Not used.

Flags: Not used.

2.2.2.16.2 NETWORKCONFIGA

The **NETWORKCONFIGA** structure defines the ANSI character version of NETWORKCONFIG.

```
typedef struct _NETWORKCONFIGA {
  LONG LanAdapter;
  DEVICENAMEA NetworkName;
  ULONG Flags;
} NETWORKCONFIGA,
 *PNETWORKCONFIGA;
```

LanAdapter: The LANA ID of the network adaptor.

NetworkName: Not used.

Flags: Not used.

2.2.2.17 ASYNCCONFIG

The asynchronous protocol driver's configuration structure.

```
#ifdef UNICODE
#define ASYNCCONFIG ASYNCCONFIGW
#define PASYNCCONFIG PASYNCCONFIGW
#else
#define ASYNCCONFIG ASYNCCONFIGA
#define PASYNCCONFIG PASYNCCONFIGA
#endif
```

2.2.2.17.1 ASYNCCONFIGW

The **ASYNCCONFIGW** structure defines the Unicode character version of ASYNCCONFIG.

```
typedef struct _ASYNCCONFIGW {
  DEVICENAMEW DeviceName;
  MODEMNAMEW ModemName;
  ULONG BaudRate;
  ULONG Parity;
  ULONG StopBits;
  ULONG ByteSize;
  ULONG fEnableDsrSensitivity  :1;
  ULONG fConnectionDriver  :1;
  FLOWCONTROLCONFIG FlowControl;
  CONNECTCONFIG Connect;
} ASYNCCONFIGW,
 *PASYNCCONFIGW;
```

DeviceName: The device's name.

ModemName: The modem's name.

BaudRate: The baud rate of the modem.

Parity: The parity setting.

StopBits: The number of stop bits.

ByteSize: The size of a byte.

fEnableDsrSensitivity: Enable Data Set Ready (DSR) sensitivity.

fConnectionDriver: Set to TRUE if there is a connection driver.

FlowControl: The flow control setting of the modem.

Connect: The connect configuration.

2.2.2.17.2 ASYNCCONFIGA

The **ASYNCCONFIGA** structure defines the ANSI character version of ASYNCCONFIG.

```
typedef struct _ASYNCCONFIGA {
  DEVICENAMEA DeviceName;
  MODEMNAMEA ModemName;
  ULONG BaudRate;
  ULONG Parity;
  ULONG StopBits;
  ULONG ByteSize;
  ULONG fEnableDsrSensitivity  :1;
  ULONG fConnectionDriver  :1;
  FLOWCONTROLCONFIG FlowControl;
  CONNECTCONFIG Connect;
} ASYNCCONFIGA,
 *PASYNCCONFIGA;
```

DeviceName: The device's name.

ModemName: The modem's name.

BaudRate: The baud rate of the modem.

Parity: The parity setting.

StopBits: The number of stop bits.

ByteSize: The size of a byte.

fEnableDsrSensitivity: Enable DSR sensitivity.

fConnectionDriver: Set to TRUE if there is a connection driver.

FlowControl: The flow control setting of the modem.

Connect: The connect configuration.

2.2.2.17.3 MODEMNAME

The name of a modem.

```
typedef WCHAR MODEMNAMEW[MODEMNAME_LENGTH + 1];

typedef WCHAR* PMODEMNAMEW;

typedef CHAR MODEMNAMEA[MODEMNAME_LENGTH + 1];

typedef CHAR* PMODEMNAMEA;
```

The following block determines **MODEMNAME**:

```
#ifdef UNICODE
#define MODEMNAME MODEMNAMEW
#define PMODEMNAME PMODEMNAMEW
#else
#define MODEMNAME MODEMNAMEA
#define PMODEMNAME PMODEMNAMEA
#endif
```

2.2.2.17.4 FLOWCONTROLCONFIG

Flow control configuration of an async driver.

```
typedef struct _FLOWCONTROLCONFIG {
  ULONG fEnableSoftwareTx  :1;
  ULONG fEnableSoftwareRx  :1;
  ULONG fEnableDTR  :1;
  ULONG fEnableRTS  :1;
  CHAR XonChar;
  CHAR XoffChar;
  FLOWCONTROLCLASS Type;
  RECEIVEFLOWCONTROLCLASS HardwareReceive;
  TRANSMITFLOWCONTROLCLASS HardwareTransmit;
} FLOWCONTROLCONFIG,
 *PFLOWCONTROLCONFIG;
```

fEnableSoftwareTx: Software transmit flow control.

fEnableSoftwareRx: Software receive flow control.

fEnableDTR: Data Terminal Ready (DTR) enabled.

fEnableRTS: Request to Send (RTS) enabled.

XonChar: Xon flow control character.

XoffChar: Xoff flow control character.

Type: The type of flow control in use.

HardwareReceive: Hardware receive flow control information.

HardwareTransmit: Hardware transmit flow control information.

2.2.2.17.4.1 FLOWCONTROLCLASS

The **FLOWCONTROLCLASS** enumeration specifies the type of flow control, if any, supported.

```
typedef  enum _FLOWCONTROLCLASS
{
  FlowControl_None,
  FlowControl_Hardware,
  FlowControl_Software
} FLOWCONTROLCLASS;
```

FlowControl_None: Flow control is not enabled.

FlowControl_Hardware: Hardware flow control is enabled.

FlowControl_Software: Software flow control is enabled.

2.2.2.17.4.2 RECEIVEFLOWCONTROLCLASS

The **RECEIVEFLOWCONTROLCLASS** enumeration specifies which, if any, means of receive flow control is supported.

```
typedef  enum _RECEIVEFLOWCONTROLCLASS
{
  ReceiveFlowControl_None,
  ReceiveFlowControl_RTS,
  ReceiveFlowControl_DTR
} RECEIVEFLOWCONTROLCLASS;
```

ReceiveFlowControl_None: No receive flow control currently.

ReceiveFlowControl_RTS: Receive flow control Request to Send (RTS).

ReceiveFlowControl_DTR: Receive flow control Data Terminal Ready (DTR).

2.2.2.17.4.3 TRANSMITFLOWCONTROLCLASS

The **TRANSMITFLOWCONTROLCLASS** enumeration specifies which, if any, means of transit flow control is supported.

```
typedef  enum _TRANSMITFLOWCONTROLCLASS
{
```

```
  TransmitFlowControl_None,
  TransmitFlowControl_CTS,
  TransmitFlowControl_DSR
} TRANSMITFLOWCONTROLCLASS;
```

TransmitFlowControl_None: No transmit flow control currently.

TransmitFlowControl_CTS: Transmit flow control Clear to Send (CTS).

TransmitFlowControl_DSR: Transmit flow control Data Set Ready (DSR).

2.2.2.17.5 CONNECTCONFIG

The **CONNECTCONFIG** structure specifies connectivity parameters.

```
typedef struct _CONNECTCONFIG {
  ASYNCCONNECTCLASS Type;
  ULONG fEnableBreakDisconnect  :1;
} CONNECTCONFIG,
 *PCONNECTCONFIG;
```

Type: Type of asynchronous connection.

fEnableBreakDisconnect: If TRUE, enable break disconnect.

2.2.2.17.5.1 ASYNCCONNECTCLASS

The **ASYNCCONNECTCLASS** enumeration is not used.

```
typedef  enum
{
  Connect_CTS,
  Connect_DSR,
  Connect_RI,
  Connect_DCD,
  Connect_FirstChar,
  Connect_Perm
} ASYNCCONNECTCLASS;
```

2.2.2.18 NASICONFIG

NASI configuration structure.

```
#ifdef UNICODE
#define NASICONFIG NASICONFIGW
#define PNASICONFIG PNASICONFIGW
#else
#define NASICONFIG NASICONFIGA
#define PNASICONFIG PNASICONFIGA
#endif /* UNICODE */
```

2.2.2.18.1 NASICONFIGW

The **NASICONFIGW** structure defines the Unicode character version of NASICONFIG.

```
typedef struct _NASICONFIGW {
  NASISPECIFICNAMEW SpecificName;
  NASIUSERNAMEW UserName;
  NASIPASSWORDW PassWord;
  NASISESIONNAMEW SessionName;
  NASIFILESERVERW FileServer;
  BOOLEAN GlobalSession;
} NASICONFIGW,
 *PNASICONFIGW;
```

SpecificName: The NASI-specific name.

UserName: The NASI user's user name.

PassWord: The NASI user's password.

SessionName: The NASI session name.

FileServer: The NASI file server name.

GlobalSession: Set to TRUE if the session is a global session.

2.2.2.18.2 NASICONFIGA

The **NASICONFIGA** structure defines the ANSI character version of NASICONFIG.

```
typedef struct _NASICONFIGA {
  NASISPECIFICNAMEA SpecificName;
  NASIUSERNAMEA UserName;
  NASIPASSWORDA PassWord;
  NASISESIONNAMEA SessionName;
  NASIFILESERVERA FileServer;
  BOOLEAN GlobalSession;
} NASICONFIGA,
 *PNASICONFIGA;
```

SpecificName: The NASI-specific name.

UserName: The NASI user's user name.

PassWord: The NASI user's password.

SessionName: The NASI session name.

FileServer: The NASI file server name.

GlobalSession: Set to TRUE if the session is a global session.

2.2.2.18.3 NASIUSERNAME

The NASI user's user name.

```
typedef WCHAR NASIUSERNAMEW[NASIUSERNAME_LENGTH + 1];
typedef CHAR NASIUSERNAMEA[NASIUSERNAME_LENGTH + 1];
```

NASIUSERNAMEW

NASIUSERNAMEA

2.2.2.18.4 NASIPASSWORD

The NASI user's password.

```
typedef WCHAR NASIPASSWORDW[NASIPASSWORD_LENGTH + 1];
typedef CHAR NASIPASSWORDA[NASIPASSWORD_LENGTH + 1];
```

2.2.2.18.5 NASISESIONNAME

The NASI session name.

```
typedef WCHAR NASISESIONNAMEW[NASISESSIONNAME_LENGTH + 1];
typedef CHAR NASISESIONNAMEA[NASISESSIONNAME_LENGTH + 1];
```

2.2.2.18.6 NASISPECIFICNAME

The NASI-specific name.

```
typedef WCHAR NASISPECIFICNAMEW[NASISPECIFICNAME_LENGTH + 1];
typedef CHAR NASISPECIFICNAMEA[NASISPECIFICNAME_LENGTH + 1];
```

2.2.2.18.7 NASIFILESERVER

The NASI file server name.

```
typedef WCHAR NASIFILESERVERW[NASIFILESERVER_LENGTH + 1];

typedef CHAR NASIFILESERVERA[NASIFILESERVER_LENGTH + 1];
```

2.2.2.19 OEMTDCONFIG

The OEM transport driver configuration structure.

```
#ifdef UNICODE
#define OEMTDCONFIG OEMTDCONFIGW
#define POEMTDCONFIG POEMTDCONFIGW
#else
#define OEMTDCONFIG OEMTDCONFIGA
#define POEMTDCONFIG POEMTDCONFIGA
#endif
```

2.2.2.19.1 OEMTDCONFIGW

The **OEMTDCONFIGW** structure defines the Unicode character version of OEMTDCONFIG.

```
typedef struct _OEMTDCONFIGW {
  LONG Adapter;
  DEVICENAMEW DeviceName;
  ULONG Flags;
} OEMTDCONFIGW,
 *POEMTDCONFIGW;
```

Adapter: The ID of the adaptor (OEM driver-specific).

DeviceName: The network name (OEM driver-specific).

Flags: Driver flags (OEM driver-specific).

2.2.2.19.2 OEMTDCONFIGA

The **OEMTDCONFIGA** structure defines the ANSI character version of OEMTDCONFIG.

```
typedef struct _OEMTDCONFIGA {
  LONG Adapter;
  DEVICENAMEA DeviceName;
  ULONG Flags;
} OEMTDCONFIGA,
 *POEMTDCONFIGA;
```

Adapter: The ID of the adaptor (OEM driver-specific).

DeviceName: The network name (OEM driver-specific).

Flags: Driver flags (OEM driver-specific).

2.2.2.20 PDCONFIG

The protocol driver configuration structure.

```
#ifdef UNICODE
#define PDCONFIG PDCONFIGW
#define PPDCONFIG PPDCONFIGW
#else
#define PDCONFIG PDCONFIGA
#define PPDCONFIG PPDCONFIGA
#endif
```

2.2.2.20.1 PDCONFIGW

The **PDCONFIGW** structure defines the Unicode character version of PDCONFIG.

```
typedef struct _PDCONFIGW {
  PDCONFIG2W Create;
  PDPARAMSW Params;
} PDCONFIGW,
 *PPDCONFIGW;
```

Create: The software configuration of the driver.

Params: The hardware configuration for the driver.

2.2.2.20.2 PDCONFIGA

The **PDCONFIGA** structure defines the ANSI character version of PDCONFIG.

```
typedef struct _PDCONFIGA {
  PDCONFIG2A Create;
  PDPARAMSA Params;
} PDCONFIGA,
 *PPDCONFIGA;
```

Create: The software configuration of the driver.

Params: The hardware configuration for the driver.

2.2.2.20.3 PDCONFIG2

The protocol driver's software configuration.

```
#ifdef UNICODE
```

```
#define PDCONFIG2 PDCONFIG2W
#define PPDCONFIG2 PPDCONFIG2W
#else
#define PDCONFIG2 PDCONFIG2A
#define PPDCONFIG2 PPDCONFIG2A
#endif
```

2.2.2.20.4 PDCONFIG2W

The **PDCONFIG2W** structure defines the Unicode character version of PDCONFIG2.

```
typedef struct _PDCONFIG2W {
  PDNAMEW PdName;
  SDCLASS SdClass;
  DLLNAMEW PdDLL;
  ULONG PdFlag;
  ULONG OutBufLength;
  ULONG OutBufCount;
  ULONG OutBufDelay;
  ULONG InteractiveDelay;
  ULONG PortNumber;
  ULONG KeepAliveTimeout;
} PDCONFIG2W,
 *PPDCONFIG2W;
```

PdName: The descriptive name of the protocol driver.

SdClass: The type of driver.

PdDLL: The driver's image name.

PdFlag: Driver flags can be any bitwise OR combination of the following values:

Value	Meaning
PD_UNUSED 0x00000001	Unused.
PD_RELIABLE 0x00000002	Error-free protocol.
PD_FRAME 0x00000004	Frame-oriented protocol.
PD_CONNECTION 0x00000008	Connection-oriented protocol.
PD_CONSOLE 0x00000010	Directly connected console.
PD_LANA 0x00000020	Network class uses LANAs (NetBIOS).

Value	Meaning
PD_TRANSPORT 0x00000040	Transport driver (lowest level).
PD_SINGLE_INST 0x00000080	Single instance only (async).
PD_NOLOW_WATERMARK 0x00000100	Low water mark to resume transmission.

OutBufLength: Optimal output buffers length.

OutBufCount: Optimal number of output buffers.

OutBufDelay: Write delay in milliseconds.

InteractiveDelay: Write delay during active input.

PortNumber: Network listen port number.

KeepAliveTimeout: Frequency to send keep-alives.

2.2.2.20.5 PDCONFIG2A

The **PDCONFIG2A** structure defines the ANSI character version of PDCONFIG2.

```
typedef struct _PDCONFIG2A {
  PDNAMEA PdName;
  SDCLASS SdClass;
  DLLNAMEA PdDLL;
  ULONG PdFlag;
  ULONG OutBufLength;
  ULONG OutBufCount;
  ULONG OutBufDelay;
  ULONG InteractiveDelay;
  ULONG PortNumber;
  ULONG KeepAliveTimeout;
} PDCONFIG2A,
 *PPDCONFIG2A;
```

PdName: The descriptive name of the protocol driver.

SdClass: The type of driver.

PdDLL: The driver's image name.

PdFlag: Driver flags can be any combination of the following values:

Value	Meaning
PD_UNUSED 0x00000001	Unused.
PD_RELIABLE	Error-free protocol.

Value	Meaning
0x00000002	
PD_FRAME 0x00000004	Frame-oriented protocol.
PD_CONNECTION 0x00000008	Connection-oriented protocol.
PD_CONSOLE 0x00000010	Directly connected console.
PD_LANA 0x00000020	Network class uses LANAs (NetBIOS).
PD_TRANSPORT 0x00000040	Transport driver (lowest level).
PD_SINGLE_INST 0x00000080	Single instance only (async).
PD_NOLOW_WATERMARK 0x00000100	Low water mark to resume transmission.

OutBufLength: Optimal output buffers length.

OutBufCount: Optimal number of output buffers.

OutBufDelay: Write delay in milliseconds.

InteractiveDelay: Write delay during active input.

PortNumber: Network listen port number.

KeepAliveTimeout: Frequency to send keep-alives.

2.2.2.20.6 PDNAME

The protocol driver name (PDNAME) data type.

```
typedef WCHAR PDNAMEW[PDNAME_LENGTH + 1];

typedef WCHAR* PPDNAMEW;

typedef CHAR PDNAMEA[PDNAME_LENGTH + 1];

typedef CHAR* PPDNAMEA;
```

The following block determines **PDNAME**:

```
#ifdef UNICODE
#define PDNAME PDNAMEW
#define PPDNAME PPDNAMEW
```

```
#else
#define PDNAME PDNAMEA
#define PPDNAME PPDNAMEA
#endif
```

2.2.2.21 SESSIONDATA

SESSIONDATA contains information about the sessions running on the Terminal Services server.

```
#define SESSIONDATA SESSIONDATAW
#define PSESSIONDATA PSESSIONDATAW
```

2.2.2.21.1 SESSIONDATAW

The **SESSIONDATAW** structure contains information about the sessions running on the Terminal Services server.

```
typedef struct _SESSIONDATAW {
  ULONG SessionId;
  WINSTATIONSTATECLASS State;
  ULONG Source;
  BOOLEAN bFullDesktop;
  GUID SessionType;
  WINSTATIONNAMEW WinStationName;
  PROTOCOLSTATUS ProtocolStatus;
} SESSIONDATAW,
 *PSESSIONDATAW;
```

SessionId: The ID of the session.

State: The current state of the session, as specified in section 2.2.1.13.

Source: Defines the type of device that connected to this particular session on the Terminal Server. It can have the following values:

Value	Meaning
1	Connected from a computer running Windows client or server operating system.
2	Connected from a device running Media Center Edition. Currently only Xbox uses this value.
3	Service Terminal.

bFullDesktop: Set to TRUE if the session is a full desktop, or FALSE otherwise.

SessionType: Describes the type of the session, as specified in section 2.2.2.4.

WinStationName: The name assigned to the session.

ProtocolStatus: The status of the protocol used to connect to the session. This is of the type PROTOCOLSTATUS, as specified in section 2.2.2.14.3.

2.2.2.22 WDCONFIG

The Winstation (session) driver configuration. This is the higher-level driver's configuration. For the Microsoft RDP implementation, this driver is rdpwd.sys.

```
#ifdef UNICODE
#define WDCONFIG WDCONFIGW
#define PWDCONFIG PWDCONFIGW
#else
#define WDCONFIG WDCONFIGA
#define PWDCONFIG PWDCONFIGA
#endif
```

2.2.2.22.1 WDCONFIGW

The **WDCONFIGW** structure defines the Unicode character version of WDCONFIG.

```
typedef struct _WDCONFIGW {
  WDNAMEW WdName;
  DLLNAMEW WdDLL;
  DLLNAMEW WsxDLL;
  ULONG WdFlag;
  ULONG WdInputBufferLength;
  DLLNAMEW CfgDLL;
  WDPREFIXW WdPrefix;
} WDCONFIGW,
 *PWDCONFIGW;
```

WdName: The descriptive name of the WinStation driver.

WdDLL: The driver's image name.

WsxDLL: The driver's user mode protocol extension DLL. Used by the termsrv.exe service to communicate with the WinStation driver and isolate termsrv from the protocol. For Microsoft's RDP implementation, this binary is rdpwsx.dll.

WdFlag: Driver flags. Can be any combination of the following values:

Value	Meaning
WDF_UNUSED 0x00000001	Not used.
WDF_SHADOW_SOURCE 0x00000002	Valid shadow source.
WDF_SHADOW_TARGET 0x00000004	Valid shadow target.
WDF_OTHER 0x00000008	Other protocol.
WDF_TSHARE 0x00000010	Microsoft RDP protocol.

Value	Meaning
WDF_DYNAMIC_RECONNECT 0x00000020	Session can resize display at reconnect. Added in Windows XP and supported from Windows XP on.
WDF_USER_VCIOCTL 0x00000040	User mode applications can send virtual channel IOCTL.
WDF_SUBDESKTOP 0x00008000	Sub-desktop session. Added in Windows XP SP2 and supported in Windows XP SP2, Windows Vista and Windows Server 2008.

WdInputBufferLength: Length of the input buffer used by the driver. Defaults to 2048.

CfgDLL: Configuration DLL used by TS administrative tools for configuration the driver. For Microsoft's RDP implementation, this binary is rdpcfgex.dll.

WdPrefix: Used as the prefix of the WinStation name generated for the sessions connected to with this WinStation driver. For Microsoft's RDP implementation, this string is "RDP".

2.2.2.22.2 WDCONFIGA

The **WDCONFIGA** structure defines the ANSI character version of WDCONFIG.

```
typedef struct _WDCONFIGA {
  WDNAMEA WdName;
  DLLNAMEA WdDLL;
  DLLNAMEA WsxDLL;
  ULONG WdFlag;
  ULONG WdInputBufferLength;
  DLLNAMEA CfgDLL;
  WDPREFIXA WdPrefix;
} WDCONFIGA,
 *PWDCONFIGA;
```

WdName: The descriptive name of the WinStation driver.

WdDLL: The driver's image name.

WsxDLL: The driver's user mode protocol extension DLL. Used by the termsrv.exe service to communicate with the WinStation driver and to isolate termsrv from the protocol. For Microsoft's RDP implementation, this binary is rdpwsx.dll.

WdFlag: Driver flags. Can be any combination of the following values:

Value	Meaning
WDF_UNUSED 0x00000001	Not used.
WDF_SHADOW_SOURCE 0x00000002	Valid shadow source.
WDF_SHADOW_TARGET 0x00000004	Valid shadow target.

Value	Meaning
WDF_OTHER 0x00000008	Other protocol.
WDF_TSHARE 0x00000010	Microsoft RDP protocol.
WDF_DYNAMIC_RECONNECT 0x00000020	Session can resize display at reconnect. Added in Windows XP and supported from Windows XP on.
WDF_USER_VCIOCTL 0x00000040	User mode applications can send virtual channel IOCTL.
WDF_SUBDESKTOP 0x00008000	Sub-desktop session. Added in Windows XP SP2 and supported in Windows XP SP2, Windows Vista and Windows Server 2008.

WdInputBufferLength: Length of the input buffer used by the driver. Defaults to 2048.

CfgDLL: Configuration DLL used by TS administrative tools for configuration the driver. For Microsoft's RDP implementation, this binary is rdpcfgex.dll.

WdPrefix: Used as the prefix of the WinStation name generated for the sessions connected to with this WinStation driver. For Microsoft's RDP implementation, this string is "RDP".

2.2.2.22.3 WDNAME

The **WDNAME** data type.

```
typedef WCHAR WDNAMEW[WDNAME_LENGTH + 1];

typedef WCHAR* PWDNAMEW;

typedef CHAR WDNAMEA[WDNAME_LENGTH + 1];

typedef CHAR* PWDNAMEA;
```

The following block determines **WDNAME**:

```
#ifdef UNICODE
#define WDNAME WDNAMEW
#define PWDNAME PWDNAMEW
#else
#define WDNAME WDNAMEA
#define PWDNAME PWDNAMEA
#endif
```

2.2.2.22.4 WDPREFIX

The **WDPREFIX** data type.

```
typedef WCHAR WDPREFIXW[WDPREFIX_LENGTH + 1];

typedef WCHAR* PWDPREFIXW;

typedef CHAR WDPREFIXA[WDPREFIX_LENGTH + 1];

typedef CHAR* PWDPREFIXA;
```

The following block determines **WDPREFIX**:

```
#ifdef UNICODE
#define WDPREFIX WDPREFIXW
#define PWDPREFIX PWDPREFIXW
#else
#define WDPREFIX WDPREFIXA
#define PWDPREFIX PWDPREFIXA
#endif
```

2.2.2.23 CDCONFIG

Connection driver configuration. Not used by the Microsoft RDP implementation. This would be used for connecting via a modem to a server.

```
#ifdef UNICODE
#define CDCONFIG  CDCONFIGW
#define PCDCONFIG PCDCONFIGW
#else
#define CDCONFIG  CDCONFIGA
#define PCDCONFIG PCDCONFIGA
#endif
```

2.2.2.23.1 CDCONFIGW

The **CDCONFIGW** structure defines the Unicode character version of CDCONFIG.

```
typedef struct _CDCONFIGW {
  CDCLASS CdClass;
  CDNAMEW CdName;
  DLLNAMEW CdDLL;
  ULONG CdFlag;
} CDCONFIGW,
 *PCDCONFIGW;
```

CdClass: Connection driver type.

CdName: Connection driver descriptive name.

CdDLL: Connection driver image name.

CdFlag: Connection driver flags. Connection driver specific.

2.2.2.23.2 CDCONFIGA

The **CDCONFIGA** structure defines the ANSI character version of CDCONFIG.

```
typedef struct CDCONFIGA {
  CDCLASS CdClass;
  CDNAMEA CdName;
  DLLNAMEA CdDLL;
  ULONG CdFlag;
} CDCONFIGA,
 *PCDCONFIGA;
```

CdClass: Connection driver type.

CdName: Connection driver descriptive name.

CdDLL: Connection driver image name.

CdFlag: Connection driver flags. Connection driver specific.

2.2.2.23.3 CDCLASS

The **CDCLASS** enumeration specifies a type of connection driver.

```
typedef  enum _CDCLASS
{
  CdNone,
  CdModem,
  CdClass_Maximum
} CDCLASS;
```

CdNone: No connection driver.

CdModem: Connection driver is a modem.

CdClass_Maximum: A given CdClass variable will always be less than this value.

2.2.2.23.4 CDNAME

Connection driver name.

```
typedef WCHAR CDNAMEW[CDNAME_LENGTH + 1];

typedef WCHAR* PCDNAMEW;

typedef CHAR CDNAMEA[CDNAME_LENGTH + 1];

typedef CHAR* PCDNAMEA;
```

The following block determines **CDNAME**:

```
#ifdef UNICODE
#define CDNAME CDNAMEW
#define PCDNAME PCDNAMEW
#else
#define CDNAME CDNAMEA
#define PCDNAME PCDNAMEA
#endif
```

2.2.2.24 WINSTATIONCREATE

Specifies a session that the user can connect to.

```
#ifdef UNICODE
#define WINSTATIONCREATE WINSTATIONCREATEW
#define PWINSTATIONCREATE PWINSTATIONCREATEW
#else
#define WINSTATIONCREATE WINSTATIONCREATEA
#define PWINSTATIONCREATE PWINSTATIONCREATEA
#endif
```

2.2.2.24.1 WINSTATIONCREATEW

The **WINSTATIONCREATEW** structure defines the Unicode character version of WINSTATIONCREATE.

```
typedef struct _WINSTATIONCREATEW {
  ULONG fEnableWinStation  :1;
  ULONG MaxInstanceCount;
} WINSTATIONCREATEW,
 *PWINSTATIONCREATEW;
```

fEnableWinStation: TRUE if enabled.

MaxInstanceCount: Maximum number of instances that can connect to the WinStation.

2.2.2.24.2 WINSTATIONCREATEA

The **WINSTATIONCREATEA** structure defines the ANSI character version of WINSTATIONCREATE.

```
typedef struct _WINSTATIONCREATEA {
  ULONG fEnableWinStation  :1;
  ULONG MaxInstanceCount;
} WINSTATIONCREATEA,
```

```
 *PWINSTATIONCREATEA;
```

fEnableWinStation: TRUE if enabled.

MaxInstanceCount: Maximum number of instances that can connect to the WinStation.

2.2.2.25 WINSTATIONCONFIG2

Specifies configuration of a session that the user can connect to.

```
#ifdef UNICODE
#define WINSTATIONCONFIG2 WINSTATIONCONFIG2W
#define PWINSTATIONCONFIG2 PWINSTATIONCONFIG2W
#else
#define WINSTATIONCONFIG2 WINSTATIONCONFIG2A
#define PWINSTATIONCONFIG2 PWINSTATIONCONFIG2A
#endif
```

2.2.2.25.1 WINSTATIONCONFIG2W

The **WINSTATIONCONFIG2W** structure defines the Unicode character version of WINSTATIONCONFIG2.

```
typedef struct _WINSTATIONCONFIG2W {
  WINSTATIONCREATEW Create;
  PDCONFIGW Pd[MAX_PDCONFIG];
  WDCONFIGW Wd;
  CDCONFIGW Cd;
  WINSTATIONCONFIGW Config;
} WINSTATIONCONFIG2W,
 *PWINSTATIONCONFIG2W;
```

Create: General creation information.

Pd: An array of protocol data configuration structures for this WinStation.

Wd: The WinStation (session) driver for this WinStation configuration.

Cd: The connection driver for this WinStation configuration.

Config: The specific configuration values for the WinStation (session).

2.2.2.25.2 WINSTATIONCONFIG2A

The **WINSTATIONCONFIG2A** structure defines the ANSI character version of WINSTATIONCONFIG2.

```
typedef struct _WINSTATIONCONFIG2A {
```

```
  WINSTATIONCREATEA Create;
  PDCONFIGA Pd[MAX_PDCONFIG];
  WDCONFIGA Wd;
  CDCONFIGA Cd;
  WINSTATIONCONFIGA Config;
} WINSTATIONCONFIG2A,
 *PWINSTATIONCONFIG2A;
```

Create: General creation information.

Pd: An array of protocol data configuration structures for this WinStation.

Wd: The WinStation (session) driver for this WinStation configuration.

Cd: The connection driver for this WinStation configuration.

Config: The specific configuration values for the WinStation (session).

2.2.2.25.3 WINSTATIONCONFIG

WinStation configuration data. Included inside a WINSTATIONCONFIG2 structure.

```
#ifdef UNICODE
#define WINSTATIONCONFIG WINSTATIONCONFIGW
#define PWINSTATIONCONFIG PWINSTATIONCONFIGW
#else
#define WINSTATIONCONFIG WINSTATIONCONFIGA
#define PWINSTATIONCONFIG PWINSTATIONCONFIGA
#endif
```

2.2.2.25.3.1 WINSTATIONCONFIGW

The **WINSTATIONCONFIGW** structure defines the Unicode character version of WINSTATIONCONFIG.

```
typedef struct _WINSTATIONCONFIGW {
  WCHAR Comment[WINSTATIONCOMMENT_LENGTH + 1];
  USERCONFIGW User;
  char OEMId[4];
} WINSTATIONCONFIGW,
 *PWINSTATIONCONFIGW;
```

Comment: The WinStation descriptive comment.

User: The user configuration data for the session (WinStation).

OEMId: The server OEM ID.

2.2.2.25.3.2 WINSTATIONCONFIGA

The **WINSTATIONCONFIGA** structure defines the ANSI character version of WINSTATIONCONFIG.

```
typedef struct _WINSTATIONCONFIGA {
  CHAR Comment[WINSTATIONCOMMENT_LENGTH + 1];
  USERCONFIGA User;
  char OEMId[4];
} WINSTATIONCONFIGA,
 *PWINSTATIONCONFIGA;
```

Comment: The WinStation descriptive comment.

User: The user configuration data for the session (WinStation).

OEMId: The server OEM ID.

2.2.2.26 POLICY_TS_MACHINE

The **POLICY_TS_MACHINE** structure defines the machine policy of the server. Each item in the policy has a flag to indicate if the policy is present and a value for the policy. Supported in Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 operating system.

```
typedef struct _POLICY_TS_MACHINE {
  ULONG fPolicyDisableClip  :1;
  ULONG fPolicyDisableCam  :1;
  ULONG fPolicyDisableCcm  :1;
  ULONG fPolicyDisableLPT  :1;
  ULONG fPolicyDisableCpm  :1;
  ULONG fPolicyPromptForPassword  :1;
  ULONG fPolicyMaxInstanceCount  :1;
  ULONG fPolicyMinEncryptionLevel  :1;
  ULONG fPolicyFipsEnabled  :1;
  ULONG fPolicyDisableAutoReconnect  :1;
  ULONG fPolicyWFProfilePath  :1;
  ULONG fPolicyWFHomeDir  :1;
  ULONG fPolicyWFHomeDirDrive  :1;
  ULONG fPolicyDenyTSConnections  :1;
  ULONG fPolicyTempFoldersPerSession  :1;
  ULONG fPolicyDeleteTempFoldersOnExit  :1;
  ULONG fPolicyColorDepth  :1;
  ULONG fPolicySessionDirectoryActive  :1;
  ULONG fPolicySessionDirectoryLocation  :1;
  ULONG fPolicySessionDirectoryClusterName  :1;
  ULONG fPolicySessionDirectoryAdditionalParams  :1;
  ULONG fPolicySessionDirectoryExposeServerIP  :1;
  ULONG fPolicyPreventLicenseUpgrade  :1;
  ULONG fPolicySecureLicensing  :1;
  ULONG fPolicyWritableTSCCPermissionsTAB  :1;
  ULONG fPolicyDisableCdm  :1;
  ULONG fPolicyForceClientLptDef  :1;
  ULONG fPolicyShadow  :1;
  ULONG fPolicyResetBroken  :1;
  ULONG fPolicyReconnectSame  :1;
  ULONG fPolicyMaxSessionTime  :1;
```

```
ULONG fPolicyMaxDisconnectionTime  :1;
ULONG fPolicyMaxIdleTime  :1;
ULONG fPolicyInitialProgram  :1;
ULONG fPolicySingleSessionPerUser  :1;
ULONG fPolicyDisableWallpaper  :1;
ULONG fPolicyKeepAlive  :1;
ULONG fPolicyEnableTimeZoneRedirection  :1;
ULONG fPolicyDisableForcibleLogoff  :1;
ULONG fPolicyLicensingMode  :1;
ULONG fPolicyExplicitLSDiscovery  :1;
ULONG fPolicyDisableTerminalServerTooltip  :1;
ULONG fDisableClip  :1;
ULONG fDisableCam  :1;
ULONG fDisableCcm  :1;
ULONG fDisableLPT  :1;
ULONG fDisableCpm  :1;
ULONG fPromptForPassword  :1;
ULONG ColorDepth  :3;
ULONG fDenyTSConnections  :1;
ULONG fTempFoldersPerSession  :1;
ULONG fDeleteTempFoldersOnExit  :1;
ULONG fWritableTSCCPermissionsTab  :1;
ULONG fDisableCdm  :1;
ULONG fForceClientLptDef  :1;
ULONG fResetBroken  :1;
ULONG fReconnectSame  :1;
ULONG fSingleSessionPerUser  :1;
ULONG fDisableWallpaper  :1;
ULONG fKeepAliveEnable  :1;
ULONG fPreventLicenseUpgrade  :1;
ULONG fSecureLicensing  :1;
ULONG fEnableTimeZoneRedirection  :1;
ULONG fDisableAutoReconnect  :1;
ULONG fDisableForcibleLogoff  :1;
ULONG fPolicyEncryptRpcTraffic  :1;
ULONG fErrorInvalidProfile  :1;
ULONG fPolicyFallbackPrintDriver  :1;
ULONG FallbackPrintDriverType  :3;
ULONG fDisableTerminalServerTooltip  :1;
BYTE bSecurityLayer;
ULONG fPolicySecurityLayer  :1;
BYTE bUserAuthentication;
ULONG fPolicyUserAuthentication  :1;
ULONG fPolicyTurnOffSingleAppMode  :1;
ULONG fTurnOffSingleAppMode  :1;
ULONG MaxInstanceCount;
ULONG LicensingMode;
BYTE MinEncryptionLevel;
WCHAR WFProfilePath[DIRECTORY_LENGTH + 1];
WCHAR WFHomeDir[DIRECTORY_LENGTH + 1];
WCHAR WFHomeDirDrive[4];
ULONG SessionDirectoryActive;
WCHAR SessionDirectoryLocation[DIRECTORY_LENGTH+1];
WCHAR SessionDirectoryClusterName[DIRECTORY_LENGTH+1];
WCHAR SessionDirectoryAdditionalParams[DIRECTORY_LENGTH+1];
ULONG SessionDirectoryExposeServerIP;
ULONG KeepAliveInterval;
SHADOWCLASS Shadow;
```

```
  ULONG MaxConnectionTime;
  ULONG MaxDisconnectionTime;
  ULONG MaxIdleTime;
  WCHAR WorkDirectory[DIRECTORY_LENGTH+1];
  WCHAR InitialProgram[INITIALPROGRAM_LENGTH + 1];
  WCHAR LicenseServers[MAX_LICENSE_SERVER_LENGTH + 1];
} POLICY_TS_MACHINE,
 *PPOLICY_TS_MACHINE;
```

fPolicyDisableClip: The policy for DisableClip is set.

fPolicyDisableCam: The policy for DisableCam is set.

fPolicyDisableCcm: The policy for DisableCcm is set.

fPolicyDisableLPT: The policy for DisableLPT is set.

fPolicyDisableCpm: The policy for DisableCpm is set.

fPolicyPromptForPassword: The policy for PromptForPassword is set.

fPolicyMaxInstanceCount: The policy for MaxInstanceCount is set.

fPolicyMinEncryptionLevel: The policy for MinEncryptionLevel is set.

fPolicyFipsEnabled: The policy for Fips is enabled. Supported from Windows Server 2003 on.

fPolicyDisableAutoReconnect: The policy for DisableAutoReconnect is set.

fPolicyWFProfilePath: The policy for WFProfilePath is set.

fPolicyWFHomeDir: The policy for WFHomeDir is set.

fPolicyWFHomeDirDrive: The policy for WFHomeDirDrive is set.

fPolicyDenyTSConnections: The policy for DenyTSConnections is set.

fPolicyTempFoldersPerSession: The policy for TempFoldersPerSession is set.

fPolicyDeleteTempFoldersOnExit: The policy for DeleteTempFoldersOnExit is set.

fPolicyColorDepth: The policy for ColorDepth is set.

fPolicySessionDirectoryActive: The policy for SessionDirectoryActive is set.

fPolicySessionDirectoryLocation: The policy for SessionDirectoryLocation is set.

fPolicySessionDirectoryClusterName: The policy for SessionDirectoryClusterName is set.

fPolicySessionDirectoryAdditionalParams: The policy for SessionDirectoryAdditionalParams is set.

fPolicySessionDirectoryExposeServerIP: The policy for SessionDirectoryExposeServerIP is set.

fPolicyPreventLicenseUpgrade: The policy for PreventLicenseUpgrade is set.

fPolicySecureLicensing: The policy for SecureLicensing is set. Supported from Windows Server 2003 on.

fPolicyWritableTSCCPermissionsTAB: The policy for WritableTSCCPermissionsTAB is set.

fPolicyDisableCdm: The policy for DisableCdm is set.

fPolicyForceClientLptDef: The policy for ForceClientLptDef is set.

fPolicyShadow: The policy for Shadow is set.

fPolicyResetBroken: The policy for ResetBroken is set.

fPolicyReconnectSame: The policy for ReconnectSame is set.

fPolicyMaxSessionTime: The policy for MaxSessionTime is set.

fPolicyMaxDisconnectionTime: The policy for MaxDisconnectionTime is set.

fPolicyMaxIdleTime: The policy for MaxIdleTime is set.

fPolicyInitialProgram: The policy for InitialProgram is set.

fPolicySingleSessionPerUser: The policy for SingleSessionPerUser is set.

fPolicyDisableWallpaper: The policy for DisableWallpaper is set.

fPolicyKeepAlive: The policy for keep-alive is set.

fPolicyEnableTimeZoneRedirection: The policy for EnableTimeZoneRedirection is set. Supported from Windows Server 2003 on.

fPolicyDisableForcibleLogoff: The policy for DisableForcibleLogoff is set. Supported from Windows Server 2003 on.

fPolicyLicensingMode: The policy for LicensingMode is set. Supported from Windows Server 2003 on.

fPolicyExplicitLSDiscovery: The policy for ExplicitLSDiscovery is set. Supported from Windows Server 2003 on.

fPolicyDisableTerminalServerTooltip: The policy for DisableTerminalServerTooltip is set. Supported from Windows Server 2003 on.

fDisableClip: Disable client clipboard redirection.

fDisableCam: Disable client audio redirection.

fDisableCcm: Disable client COM port redirection.

fDisableLPT: Disable client LPT port redirection.

fDisableCpm: Disable client printer redirection.

fPromptForPassword: Set to FALSE to log on user with previously provided credentials, or TRUE to prompt the user for password.

ColorDepth: The color depth of the session. The supported values 1, 2, 4, 8, 16 are translated, respectively, as the following number of colors supported: 16 (4 bpp), 256 (8 bpp), 65,536

(16 bpp), 16 million (24 bpp), 32,768 (15 bpp). On Windows Vista and Windows Server 2008 the value 32 was added to indicate 32-bit color.

fDenyTSConnections: If set to TRUE, TS is effectively disabled since remote connections will be declined.

fTempFoldersPerSession: Set to TRUE if there are temporary folders per session instead of one common temp folder, or FALSE otherwise.

fDeleteTempFoldersOnExit: Delete temporary folders on session exit.

fWritableTSCCPermissionsTab: If set to TRUE, Administrators can change the per-connection security description, or FALSE otherwise.

fDisableCdm: Disable client drive redirection.

fForceClientLptDef: Force the client's redirected printer to be the default printer for the user.

fResetBroken: Reset the session if the connection is broken or if the connection or idle timers expire.

fReconnectSame: Set to FALSE to indicate that the user can reconnect from any client computer to a disconnected session. TRUE indicates that the user can only reconnect to a disconnected session from the same client computer that initially established the disconnected session. Logging on from a different client computer will lead to a new TS session being created.

fSingleSessionPerUser: Each user can have only a single session.

fDisableWallpaper: Display of the desktop wallpaper in the session has been disabled.

fKeepAliveEnable: Keep-alive is enabled.

fPreventLicenseUpgrade: Licenses are prevented from being upgraded.

fSecureLicensing: Secure licensing is enabled. Supported from Windows Server 2003 on.

fEnableTimeZoneRedirection: Client time zone redirection is enabled. Supported from Windows Server 2003 on.

fDisableAutoReconnect: Disable auto-reconnect functionality.

fDisableForcibleLogoff: Disable forcible logoff. Supported from Windows Server 2003 on.

fPolicyEncryptRpcTraffic: Policy for EncryptRpcTraffic is set. Supported from Windows Server 2003 on.

fErrorInvalidProfile: Set to TRUE if WFProfilePath, WFHomeDir or WFHomeDirDrive is invalid (too long), or FALSE otherwise. Supported from Windows Server 2003 on.

fPolicyFallbackPrintDriver: The policy for FallbackPrintDriver is set. Supported from Windows Server 2003 on.

FallbackPrintDriverType: The fallback printer driver type. Can be NO_FALLBACK_DRIVERS, FALLBACK_BESTGUESS, FALLBACK_PCL, FALLBACK_PS, FALLBACK_PCLANDPS. Supported from Windows Server 2003 on.

fDisableTerminalServerTooltip: Disable terminal server tooltip. Supported from Windows Server 2003 on.

bSecurityLayer: If non-zero, indicates the SSL security layer in use. Supported from Windows Server 2003 on.

fPolicySecurityLayer: The policy for SecurityLayer is set. Supported from Windows Server 2003 on.

bUserAuthentication: The user authentication level. Supported from Windows Vista on. Can be any of the following values.

- TS_USER_AUTHENTICATION_NONE
- TS_USER_AUTHENTICATION_VIA_HYBRID
- TS_USER_AUTHENTICATION_VIA_SSL
- TS_USER_AUTHENTICATION_DEFAULT (same as TS_USER_AUTHENTICATION_NONE)

fPolicyUserAuthentication: The policy for UserAuthentication is set. Supported from Windows Vista on.

fPolicyTurnOffSingleAppMode: The policy for TurnOffSingleAppMode is set. Supported from Windows Server 2003 on.

fTurnOffSingleAppMode: TRUE specifies the desktop is always displayed when a client connects to a remote computer. FALSE specifies an initial program can be specified that runs on the remote computer after the client connects to the remote computer.

MaxInstanceCount: The maximum number of instances that can connect.

LicensingMode: The licensing mode of the server.

MinEncryptionLevel: The minimum allowed encryption level.

WFProfilePath: The TS profile path. Overrides standard profile path.

WFHomeDir: The TS home directory path. Overrides standard home directory.

WFHomeDirDrive: The TS home directory drive. Overrides standard home directory.

SessionDirectoryActive: Set to TRUE if the machine is part of a Session Directory farm, or FALSE otherwise.

SessionDirectoryLocation: The name of the Session Directory Server.

SessionDirectoryClusterName: The name of the Session Directory farm that this machine belongs to.

SessionDirectoryAdditionalParams: Additional parameters to pass to the session directory. This is an opaque type.

SessionDirectoryExposeServerIP: If set to TRUE, expose the server's IP address to the client; otherwise FALSE.

KeepAliveInterval: Specifies the interval between keep-alives.

Shadow: Specifies whether shadowing of the session is allowed.

MaxConnectionTime: The maximum allowed session connection time setting of the session.

MaxDisconnectionTime: The maximum allowed session disconnect time of the session.

MaxIdleTime: The maximum allowed session idle time setting of the session.

WorkDirectory: The work directory for the initial program.

InitialProgram: The program to run instead of explorer.exe, if set.

LicenseServers: A hardcoded array of license servers the server will use instead of using license server discovery.

2.2.2.27 WINSTATIONUSERTOKEN

The **WINSTATIONUSERTOKEN** structure defines the user token for a session.

```
typedef struct _WINSTATIONUSERTOKEN {
  HANDLE ProcessId;
  HANDLE ThreadId;
  HANDLE UserToken;
} WINSTATIONUSERTOKEN,
 *PWINSTATIONUSERTOKEN;
```

ProcessId: Specifies the Process ID.

ThreadId: Specifies the handle to the calling thread.

UserToken: Return the user token that is currently logged on to the session.

2.2.2.28 WINSTATIONVIDEODATA

The **WINSTATIONVIDEODATA** structure defines the resolution and color depth of a session.

```
typedef struct _WINSTATIONVIDEODATA {
  USHORT HResolution;
  USHORT VResolution;
  USHORT fColorDepth;
} WINSTATIONVIDEODATA,
 *PWINSTATIONVIDEODATA;
```

HResolution: Specifies the horizontal resolution, in pixels.

VResolution: Specifies the vertical resolution, in pixels.

fColorDepth: Specifies the color depth. The supported values 1, 2, 4, 8, and 16 are translated, respectively, as the following number of colors supported: 16 (4 bpp), 256 (8 bpp), 65,536 (16 bpp), 16 million (24 bpp), 32,768 (15 bpp). On Windows Vista and Windows Server 2008 the value 32 was added to indicate 32-bit color.

2.2.2.29 WINSTATIONLOADINDICATORDATA

The **WINSTATIONLOADINDICATORDATA** structure defines data used for the load balancing of a server.

```
typedef struct _WINSTATIONLOADINDICATORDATA {
  ULONG RemainingSessionCapacity;
  LOADFACTORTYPE LoadFactor;
  ULONG TotalSessions;
  ULONG DisconnectedSessions;
  LARGE_INTEGER IdleCPU;
  LARGE_INTEGER TotalCPU;
  ULONG RawSessionCapacity;
  ULONG reserved[9];
} WINSTATIONLOADINDICATORDATA,
 *PWINSTATIONLOADINDICATORDATA;
```

RemainingSessionCapacity: The estimated number of additional sessions that can be supported given the CPU constraint.

LoadFactor: Indicates the most constrained current resource.

TotalSessions: The total number of sessions.

DisconnectedSessions: The number of disconnected sessions.

IdleCPU: The idle CPU time.

TotalCPU: The user and kernel mode CPU time.

RawSessionCapacity: The raw number of sessions capacity.

reserved: Reserved.

2.2.2.29.1 LOADFACTORTYPE

The **LOADFACTORTYPE** enumeration specifies the most constrained resource affecting load balancing.

```
typedef  enum _LOADFACTORTYPE
{
  ErrorConstraint,
  PagedPoolConstraint,
  NonPagedPoolConstraint,
  AvailablePagesConstraint,
  SystemPtesConstraint,
  CPUConstraint
} LOADFACTORTYPE;
```

ErrorConstraint: An error occurred while obtaining constraint data.

PagedPoolConstraint: The amount of paged pool is the constraint.

NonPagedPoolConstraint: The amount of non-paged pool is the constraint.

AvailablePagesConstraint: The amount of available pages is the constraint.

SystemPtesConstraint: The number of system page table entries (PTEs) is the constraint.

CPUConstraint: CPU usage is the constraint.

2.2.2.30 WINSTATIONSHADOW

The **WINSTATIONSHADOW** structure is used for WinStationQueryInformation/WinQuerySetInformation operations.

```
typedef struct _WINSTATIONSHADOW {
  SHADOWSTATECLASS ShadowState;
  SHADOWCLASS ShadowClass;
  ULONG SessionId;
  ULONG ProtocolType;
} WINSTATIONSHADOW,
 *PWINSTATIONSHADOW;
```

ShadowState: Specifies the current state of shadowing.

ShadowClass: Specifies the type of shadowing.

SessionId: Specifies the session ID of the session.

ProtocolType: Specifies the type of protocol on the session. Can be the following values:

Name	Value
PROTOCOL_OTHERS	1
PROTOCOL_RDP	2
PROTOCOL_CONSOLE	3<51>

2.2.2.30.1 SHADOWSTATECLASS

The **SHADOWSTATECLASS** enumeration specifies WinStation shadow states.

```
typedef  enum _SHADOWSTATECLASS
{
  State_NoShadow = 0,
  State_Shadowing = 1,
  State_Shadowed = 2
} SHADOWSTATECLASS;
```

State_NoShadow: No shadow operations are currently being performed on this session.

State_Shadowing: The session is shadowing a different session. The current session is referred to as a shadow client.

State_Shadowed: The session is being shadowed by a different session. The current session is referred to as a shadow target.

2.2.2.31 WINSTATIONPRODID

Represents a product ID for the session.

```
#ifdef UNICODE
#define WINSTATIONPRODID WINSTATIONPRODIDW
#define PWINSTATIONPRODID PWINSTATIONPRODIDW
#else
#define WINSTATIONPRODID WINSTATIONPRODIDA
#define PWINSTATIONPRODID PWINSTATIONPRODIDA
#endif
```

2.2.2.31.1 WINSTATIONPRODIDW

The **WINSTATIONPRODIDW** structure defines the Unicode character version of WINSTATIONPRODID.

```
typedef struct _WINSTATIONPRODIDW {
  WCHAR DigProductId[CLIENT_PRODUCT_ID_LENGTH];
  WCHAR ClientDigProductId[CLIENT_PRODUCT_ID_LENGTH];
  WCHAR OuterMostDigProductId[CLIENT_PRODUCT_ID_LENGTH];
  ULONG curentSessionId;
  ULONG ClientSessionId;
  ULONG OuterMostSessionId;
} WINSTATIONPRODIDW,
 *PWINSTATIONPRODIDW;
```

DigProductId: The product ID of the server.

ClientDigProductId: The product ID of the client.

OuterMostDigProductId: Not used.

curentSessionId: The current session identifier.

ClientSessionId: The client's session identifier.

OuterMostSessionId: Not used.

2.2.2.31.2 WINSTATIONPRODIDA

The **WINSTATIONPRODIDA** structure defines the ANSI character version of WINSTATIONPRODID.

```
typedef struct _WINSTATIONPRODIDA {
  CHAR DigProductId[CLIENT_PRODUCT_ID_LENGTH];
  CHAR ClientDigProductId[CLIENT_PRODUCT_ID_LENGTH];
  CHAR OuterMostDigProductId[CLIENT_PRODUCT_ID_LENGTH];
  ULONG curentSessionId;
  ULONG ClientSessionId;
  ULONG OuterMostSessionId;
} WINSTATIONPRODIDA,
```

```
*PWINSTATIONPRODIDA;
```

DigProductId: The product ID of the server.

ClientDigProductId: The product ID of the client.

OuterMostDigProductId: Not used.

curentSessionId: The current session identifier.

ClientSessionId: The client's session identifier.

OuterMostSessionId: Not used.

2.2.2.32 WINSTATIONREMOTEADDRESS

The **WINSTATIONREMOTEADDRESS** structure specifies the client's remote address. Only TCP/IP addresses are supported.<52>

```
typedef struct {
  unsigned short sin_family;
  union {
    struct {
      USHORT sin_port;
      ULONG in_addr;
      UCHAR sin_zero[8];
    } ipv4;
    struct {
      USHORT sin6_port;
      ULONG sin6_flowinfo;
      USHORT sin6_addr[8];
      ULONG sin6_scope_id;
    } ipv6;
  };
} WINSTATIONREMOTEADDRESS,
 *PWINSTATIONREMOTEADDRESS;
```

sin_family: MUST be AF_INET to indicate IPv4 is supported or AF_INET6 to indicate IPv6 is supported.

ipv4: IPv4 address; for more information, see [MSDN-TDIADDRESS].

sin_port: Specifies a TCP or UDP port number.

in_addr: Indicates the IP address.

sin_zero: An array filled with zeros.

ipv6: IPv6 address. Structure SOCKADDR_IN6's fields are identical to the fields in this field.

sin6_port: Specifies a TCP or UDP port number.

sin6_flowinfo: Ipv6 flow information.

sin6_addr: Indicates the IP address.

sin6_scope_id: Set of interfaces for a scope.

2.2.2.33 ExtendedClientCredentials

The **ExtendedClientCredentials** structure holds longer user name, password, and domain fields.

```
typedef struct _ExtendedClientCredentials {
  WCHAR UserName[EXTENDED_USERNAME_LEN + 1];
  WCHAR Password[EXTENDED_PASSWORD_LEN + 1];
  WCHAR Domain[EXTENDED_DOMAIN_LEN + 1];
} ExtendedClientCredentials,
 *pExtendedClientCredentials;
```

UserName: Specifies the user's username.

Password: Specifies the user's password.

Domain: Specifies the user's domain name.

2.2.2.34 TS_TRACE

The **TS_TRACE** structure specifies fields used for configuring tracing operations in TS binaries if they are checked.

```
typedef struct _TS_TRACE {
  WCHAR TraceFile[256];
  BOOLEAN fDebugger;
  BOOLEAN fTimestamp;
  ULONG TraceClass;
  ULONG TraceEnable;
  WCHAR TraceOption[64];
} TS_TRACE,
 *PTS_TRACE;
```

TraceFile: Specifies the file name, if any, to write debug information to.

fDebugger: Specifies whether debugger is attached.

fTimestamp: Specifies whether to append time stamp to the traces logged.

TraceClass: Classes of tracing to log. They enable tracing for the various Terminal Server binaries/functionalities.

Value	Meaning
TC_ICASRV 0x00000001	Termsrv service

Value	Meaning
TC_ICAAPI 0x00000002	icaapi.dll
TC_ICADD 0x00000004	termdd.sys
TC_WD 0x00000008	winstation driver (rdpwd.sys for TS RDP sessions)
TC_CD 0x00000010	connection driver (not used by Terminal Services)
TC_PD 0x00000020	protocol driver (rdpwd.sys for TS RDP sessions)
TC_TD 0x00000040	transport driver (tdtcp.sys or tdpipe.sys for TS RDP sessions)
TC_RELIABLE 0x00000100	reliable protocol driver (not used by Terminal Services)
TC_FRAME 0x00000200	frame protocol driver (not used by Terminal Services)
TC_COMP 0x00000400	compression (not used by Terminal Services)
TC_CRYPT 0x00000800	encryption (not used by Terminal Services)
TC_TW 0x10000000	thinwire (not used by Terminal Services)
TC_DISPLAY 0x10000000	display driver (rdpdd.dll for TS RDP sessions)
TC_WFSHELL 0x20000000	wfshell (not used by Terminal Services)
TC_WX 0x40000000	winstation extension (rdpwsx.dll for TS RDP sessions)
TC_LOAD 0x80000000	load balancing (not used by Terminal Services)
TC_ALL 0xffffffff	everything

TraceEnable: Type of tracing calls log.

Value	Meaning
TT_API1 0x00000001	api level 1
TT_API2 0x00000002	api level 2

Value	Meaning
TT_API3 0x00000004	api level 3
TT_API4 0x00000008	api level 4
TT_OUT1 0x00000010	output level 1
TT_OUT2 0x00000020	output level 2
TT_OUT3 0x00000040	output level 3
TT_OUT4 0x00000080	output level 4
TT_IN1 0x00000100	input level 1
TT_IN2 0x00000200	input level 2
TT_IN3 0x00000400	input level 3
TT_IN4 0x00000800	input level 4
TT_ORAW 0x00001000	raw output data
TT_IRAW 0x00002000	raw input data
TT_OCOOK 0x00004000	cooked output data
TT_ICOOK 0x00008000	cooked input data
TT_SEM 0x00010000	semaphores
TT_NONE 0x10000000	only view errors
TT_ERROR 0xffffffff	error condition

TraceOption: Trace option string.

2.2.2.35 BEEPINPUT

The **BEEPINPUT** structure performs a beep in the session.

```
typedef struct _BEEPINPUT {
  ULONG uType;
} BEEPINPUT,
 *PBEEPINPUT;
```

uType: If the session ID is 0, this can be any of the values that can be passed to the standard MessageBeep. If session ID is not 0, a frequency and duration is chosen by the server to send as a beep to the session.

2.2.2.36 WINSTATIONCLIENTDATA

The **WINSTATIONCLIENTDATA** structure is a ClientData structure sent through RpcWinStationSetInformation to send data to the client.

```
typedef struct _WINSTATIONCLIENTDATA {
  CLIENTDATANAME DataName;
  BOOLEAN fUnicodeData;
} WINSTATIONCLIENTDATA,
 *PWINSTATIONCLIENTDATA;
```

DataName: Identifies the type of data sent in this WINSTATIONCLIENTDATA structure. The definition is dependent on the caller and on the client receiving it. This can be CLIENTDATA_SERVER, CLIENTDATA_USERNAME, CLIENTDATA_DOMAIN or a third-party data name following a similar format to the CLIENTDATANAME data type.

fUnicodeData: Data is in Unicode format.

2.2.2.37 LCPOLICYINFO_V1

LCPOLICYINFO_V1 contains information about the licensing policy used on a Terminal Server. This is returned when version 1 of the policy is used. 1 is the only version of licensing policies currently supported.

```
#ifdef UNICODE
#define LCPOLICYINFO_V1 LCPOLICYINFO_V1W
#define LPLCPOLICYINFO_V1 LPLCPOLICYINFO_V1W
#else
#define LCPOLICYINFO_V1 LCPOLICYINFO_V1A
#define LPLCPOLICYINFO_V1 LPLCPOLICYINFO_V1A
#endif
```

2.2.2.37.1 LCPOLICYINFO_V1W

The **LCPOLICYINFO_V1W** structure defines the Unicode character version of LCPOLICYINFO_V1.

```
typedef struct {
  ULONG ulVersion;
  WCHAR* lpPolicyName;
```

```
  WCHAR* lpPolicyDescription;
} LCPOLICYINFO_V1W,
 *LPLCPOLICYINFO_V1W;
```

ulVersion: The version of the policy.

lpPolicyName: The name of the policy.

lpPolicyDescription: The description of the policy.

2.2.2.37.2 LCPOLICYINFO_V1A

The **LCPOLICYINFO_V1A** structure defines the ANSI character version of LCPOLICYINFO_V1.

```
typedef struct {
  ULONG ulVersion;
  char* lpPolicyName;
  char* lpPolicyDescription;
} LCPOLICYINFO_V1A,
 *LPLCPOLICYINFO_V1A;
```

ulVersion: The version of the policy.

lpPolicyName: The name of the policy.

lpPolicyDescription: The description of the policy.

2.2.2.38 SESSION_CHANGE

The **SESSION_CHANGE** structure contains the ID of a session running on a Terminal Server and a mask of the notifications that were received for that session.

```
typedef struct _SESSION_CHANGE {
  LONG SessionId;
  TNotificationId NotificationId;
} SESSION_CHANGE,
 *PSESSION_CHANGE;
```

SessionId: Identifies the session for which notification was received.

NotificationId: Mask of the notifications that were received for this session.

2.2.2.39 RCM_REMOTEADDRESS

The **RCM_REMOTEADDRESS** structure defines a remote address.

```
typedef struct {
     union switch (unsigned short sin_family) u {
```

```
        case 2:
             struct {
                  USHORT sin_port;
                  ULONG  in_addr;
                  UCHAR  sin_zero[8];
             } ipv4;
        case 23:
             struct {
                  USHORT sin6_port;
                  ULONG  sin6_flowinfo;
                  USHORT sin6_addr[8];
                  ULONG  sin6_scope_id;
             } ipv6;
        };
  } RCM_REMOTEADDRESS, *PRCM_REMOTEADDRESS;
```

ipv4: IPv4 address. For more information, see [MSDN-TDIADDRESS].

sin_port: Specifies a TCP or UDP port number.

in_addr: Indicates the IP address.

sin_zero: An array filled with zeros.

ipv6: Ipv6 address.

sin6_port: Specifies a TCP or UDP port number.

sin6_flowinfo: Ipv6 flow information.

sin6_addr: Indicates the IP address.

sin6_scope_id : Set of interfaces for a scope.

2.2.2.40 CLIENT_STACK_ADDRESS

The **CLIENT_STACK_ADDRESS** structure represents the client network address.

```
typedef struct _SESSION_CHANGE {
  BYTE Address[STACK_ADDRESS_LENGTH];
} CLIENT_STACK_ADDRESS,
 *PCLIENT_STACK_ADDRESS;
```

Address: The first two bytes represent the address family that the client network address belongs to (for more information, see [MSDN-SOCKET]) and the remaining bytes represents the client network address in a TDI_ADDRESS_IP structure (for more information, see [MSDN-TDIADDRESS]).

3 Protocol Details

The methods comprising this RPC interface all return 0x00000000 on success and a non-zero, implementation-specific, error code on failure. Unless otherwise specified below, a server-side implementation of this protocol may choose any non-zero Win32 error value to signify an error condition, as specified in section 1.8.

The client side of the Terminal Server Runtime Interface (RPC) Protocol MUST NOT interpret returned error codes. The client side of the Terminal Server Runtime Interface (RPC) Protocol MUST simply return error codes to the invoking application without taking any protocol action.

Note that the terms "client side" and "server side" refer to the initiating and receiving ends of the protocol, respectively, rather than to client or server versions of an operating system. These methods MUST all behave the same regardless of whether the "server side" of the protocol is running in a client or server version of an operating system.

3.1 Local Session Manager Client Details

3.1.1 Abstract Data Model

No abstract data model is required.

3.1.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE].

3.1.3 Initialization

The client MUST create an RPC connection to the Terminal Server by using the details specified in section 2.1.

3.1.4 Message Processing Events and Sequencing Rules

This is a stateless protocol. No sequence of method calls is imposed on this protocol.

When a method completes, the values returned by RPC MUST be returned unmodified to the upper layer.

The client MUST ignore errors returned from the RPC server and notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.

3.1.5 Timer Events

There are no timer events.

3.1.6 Other Local Events

There are no local events.

3.2 Local Session Manager Server Details

3.2.1 Abstract Data Model

No abstract data model is required.

3.2.2 Timers

No timers are required by this protocol.

3.2.3 Initialization

Parameters necessary to initialize the RPC protocol are specified in section 2.1.

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. For more information, see [MS-RPCE] section 1.3.

All methods implemented by the TS server SHOULD enforce appropriate security measures to make sure that the TS client has required permissions to execute the method. All methods MUST be RPC asynchronous calls.

The methods MAY throw an exception and the TS client MUST gracefully handle these exceptions.

3.2.4.1 TermSrvSession Methods

TermSrvSession provides methods that manage, and provide information about, a session on a given Terminal Server. The version for this interface is 1.0.

For information about endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcOpenSession	Returns a handle to a specified session on the Terminal Server. Opnum: 0
RpcCloseSession	Closes the connection to the specified session on the Terminal Server. Opnum: 1
RpcConnect	Reconnects a session handle returned by **RpcOpenSession** to another specified session on the Terminal Server. Opnum: 2
RpcDisconnect	Disconnects the specified session on the Terminal Server. Opnum: 3
RpcLogoff	Logs off the specified session on the Terminal. Opnum: 4
RpcGetUserName	Gets the username and domain name of the user logged on to the specified session on the Terminal Server.

Method	Description
	Opnum: 5
RpcGetTerminalName	Gets the name of the terminal associated with the specified session on the Terminal Server. Opnum: 6
RpcGetState	Gets state of the specified session on the Terminal Server. Opnum: 7
RpcIsSessionDesktopLocked	Checks whether the specified session on the Terminal Server is locked. Opnum: 8
RpcShowMessageBox	Displays a message box, with a specified message and title, in the target user session running on the Terminal Server. Opnum: 9
RpcGetTimes	Gets the connected, disconnected, and logged on time for the specified session on the Terminal Server. Opnum: 10
RpcGetSessionCounters	Returns the various performance counters associated with the Terminal Server. Opnum: 11
RpcGetSessionInformation	Retrieves information about a specified session running on a Terminal Server. Opnum: 12
RpcSwitchToServicesSession	Retrieves the current session opened by the RPC client and connects it to the services session (Session Id 0) as specified in [MSDN-SCWV] on the Terminal Server. Opnum: 13
RpcRevertFromServicesSession	Disconnects the client from the services session (Session Id 0) as specified in [MSDN-SCWV] and connects it back to the previous session on the Terminal Server. Opnum: 14
RpcGetLoggedOnCount	Gets the number of user-connected and device-connected sessions. Opnum: 15

3.2.4.1.1 RpcOpenSession (Opnum 0)

The **RpcOpenSession** method returns a handle to a specified session on the Terminal Server.

```
HRESULT RpcOpenSession(
  [in] handle_t hBinding,
  [in] LONG SessionId,
  [out] SESSION_HANDLE* phSession
);
```

hBinding: The RPC binding handle (for more information, see [MSDN-RPCBIND]).

SessionId: The identifier of the session to open. This session MUST be present on the Terminal Server, or this call will fail.

phSession: A handle to the session. This is of type **SESSION_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.2 RpcCloseSession (Opnum 1)

The **RpcCloseSession** method closes the connection to the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcCloseSession(
  [in, out] SESSION_HANDLE_EXCLUSIVE* phSession
);
```

phSession: Pointer to a handle to the session to close which was returned by **RpcOpenSession**. This is of type **SESSION_HANDLE_EXCLUSIVE** which has the same definition as **SESSION_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.3 RpcConnect (Opnum 2)

The **RpcConnect** method reconnects a session handle returned by **RpcOpenSession** to another specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**. The caller MUST be either the same as the one who is logged on to the target session or logged on as System.

```
HRESULT RpcConnect(
  [in] SESSION_HANDLE hSession,
  [in] LONG TargetSessionId,
  [in, string] WCHAR* szPassword
);
```

hSession: The handle to a session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

TargetSessionId: The identifier of the session on the Terminal Server to reconnect the session handle to. This session MUST be present on the Terminal Server or this call will fail.

szPassword: The password of the user connected to the current session. This is an optional field. If not specified, the TS server will impersonate the current user making the call and check if it has permission to disconnect the current session.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.4 RpcDisconnect (Opnum 3)

The **RpcDisconnect** method disconnects the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**. The caller must have the right to disconnect the session.

```
HRESULT RpcDisconnect(
  [in] SESSION_HANDLE hSession
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.5 RpcLogoff (Opnum 4)

The **RpcLogoff** method logs off the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**. The caller must have the right to log off the session.

```
HRESULT RpcLogoff(
  [in] SESSION_HANDLE hSession
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.6 RpcGetUserName (Opnum 5)

The **RpcGetUserName** method gets the username and domain name of the user logged on to the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcGetUserName(
  [in] SESSION_HANDLE hSession,
  [out, string] WCHAR** pszUserName,
  [out, string] WCHAR** pszDomain
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

pszUserName: The name of the user who is logged on to the specific session.

pszDomain: The domain to which the currently logged-in user belongs.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.7 RpcGetTerminalName (Opnum 6)

The **RpcGetTerminalName** method gets the name of the terminal associated with the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcGetTerminalName(
  [in] SESSION_HANDLE hSession,
  [out, string] WCHAR** pszTerminalName
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

pszTerminalName: The name of the terminal associated with the specific session.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000	Successful completion.

Return value/code	Description
S_OK	

3.2.4.1.8 RpcGetState (Opnum 7)

The **RpcGetState** method gets the state of the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcGetState(
  [in] SESSION_HANDLE hSession,
  [out] LONG* plState
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

plState: Defines the current state of the session. The following are the possible values:

Value	Meaning
Active 0	The user logged on to the session.
Connected 1	The session connected to the client.
ConnectQuery 2	The session is in the process of connecting to the client.
Shadow 3	The session is shadowing another session.
Disconnected 4	The session logged on without client.
Idle 5	The session is waiting for client to connect.
Listen 6	The session is listening for Terminal Server connections.
Reset 7	The session is being reset.
Down 8	The session is down due to some error condition.
Init 9	The session is being initialized.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.9 RpcIsSessionDesktopLocked (Opnum 8)

The **RpcIsSessionDesktopLocked** method checks if the specified session on the Terminal Server is in locked state or not. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcIsSessionDesktopLocked(
  [in] SESSION_HANDLE hSession
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) if the session is locked; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.10 RpcShowMessageBox (Opnum 9)

The **RpcShowMessageBox** method displays a message box, with specified message and title, in the target user session running on the Terminal Server. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcShowMessageBox(
  [in] SESSION_HANDLE hSession,
  [in, string] WCHAR* szTitle,
  [in, string] WCHAR* szMessage,
  [in] ULONG ulStyle,
  [in] ULONG ulTimeout,
  [out] ULONG* pulResponse,
  [in] BOOL bDoNotWait
);
```

hSession: The handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

szTitle: The title to assign to the message box.

szMessage: The message to display inside the message box.

ulStyle: Specifies the contents and behavior of the message box. This value is typically MB_OK. For a complete list of values, see the **uType** parameter of the MessageBox function. For more information, see [MSDN-MSGBOX].

ulTimeout: The time for which to display the message box.

pulResponse: Pointer to a variable that receives the user's response, which can be one of the following values defined in [MSDN-MSGBOX].

Value	Meaning
IDABORT 3	The Abort button was selected.
IDCANCEL 2	The Cancel button was selected.
IDIGNORE 5	The Ignore button was selected.
IDNO 7	The No button was selected.
IDOK 1	The OK button was selected.
IDRETRY 4	The Retry button was selected.
IDYES 6	The Yes button was selected.
IDASYNC 32001	The *bDoNotWait*parameter was TRUE, so the function returned without waiting for a response.
IDTIMEOUT 32000	The *bDoNotWait* parameter was FALSE and the time-out interval elapsed.

bDoNotWait: Set to TRUE to wait for the message box to timeout or close, or false otherwise.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.11 RpcGetTimes (Opnum 10)

The **RpcGetTimes** method gets the connected, disconnected, and logged-on time for the specified session on the Terminal Server. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcGetTimes(
  [in] SESSION_HANDLE hSession,
  [out] hyper* pConnectTime,
  [out] hyper* pDisconnectTime,
  [out] hyper* pLogonTime
);
```

hSession: Handle to the session returned by **RpcOpenSession**. This is of type **SESSION_HANDLE**.

pConnectTime: The time the session was last connected to.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

pDisconnectTime: The time the session was last disconnected from.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

pLogonTime: The time the session was last logged on to.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.12 RpcGetSessionCounters (Opnum 11)

The **RpcGetSessionCounters** method returns the various performance counters associated with the Terminal Server.

```
HRESULT RpcGetSessionCounters(
  [in] handle_t hBinding,
  [in, out, size_is(uEntries)] PTS_COUNTER pCounter,
  [in] ULONG uEntries
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

pCounter: An array of **PTS_COUNTER** structures, each containing a counter and its value. There will be **uEntries** number of counters in the array.

uEntries: The number of entries in the *pCounter* array.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.13 RpcGetSessionInformation (Opnum 12)

The **RpcGetSessionInformation** method retrieves information about a specified session running on a Terminal Server.

```
HRESULT RpcGetSessionInformation(
  [in] handle_t hBinding,
  [in] LONG SessionId,
```

```
  [ref, out] PLSMSESSIONINFORMATION pSessionInfo
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The identifier of the session whose information is to be retrieved.

pSessionInfo: A **PLSMSESSIONINFORMATION** element containing information about the session.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.14 RpcSwitchToServicesSession (Opnum 13)

The **RpcSwitchToServicesSession** method retrieves the current session opened by the RPC client and connects it to the services session (Session Id 0) on the Terminal Server. For more information, see [MSDN-SCWV]. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcSwitchToServicesSession(
  [in] handle_t hBinding
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.15 RpcRevertFromServicesSession (Opnum 14)

The **RpcRevertFromServicesSession** method disconnects the client from the services session (Session Id 0) and connects it back to the previous session on the Terminal Server. For more information, see [MSDN-SCWV]. This method MUST be called after **RpcOpenSession** and **RpcSwitchToServicesSession**.

```
HRESULT RpcRevertFromServicesSession(
  [in] handle_t hBinding
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.1.16 RpcGetLoggedOnCount (Opnum 15)

The **RpcGetLoggedOnCount** method gets the number of user-connected and device-connected sessions. This method MUST be called after **RpcOpenSession**.

```
HRESULT RpcGetLoggedOnCount(
  [in] handle_t hBinding,
  [out] ULONG* pUserSessions,
  [out] ULONG* pDeviceSessions
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

pUserSessions: The number of connected user sessions.

pDeviceSessions: The number of device-connected sessions.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.2 TermSrvNotification

The **TermSrvNotification** or LSM Notification interface, provides methods that manage asynchronous operations. Methods that initiate asynchronous operations return a pointer to an LSM Notification (TermSrvNotification) interface, allowing the caller to optionally cancel, or wait for, the status of the asynchronous operation. The version for this interface is 1.0.

For information about endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcWaitForSessionState	Blocks until the state of the specified session running on a Terminal Server changes to the desired state. Opnum: 0
RpcRegisterAsyncNotification	Registers for an event or events happening on a Terminal Server. Opnum: 1
RpcWaitAsyncNotification	Starts the wait for the specified notification to be signaled by the

Method	Description
	Terminal Server. Opnum: 2
RpcUnRegisterAsyncNotification	Cancels the asynchronous operation of waiting for notification from the Terminal Server. Opnum: 3

3.2.4.2.1 RpcWaitForSessionState (Opnum 0)

The **RpcWaitForSessionState** method blocks until the state of the specified session running on a Terminal Server changes to the desired state.

```
HRESULT RpcWaitForSessionState(
  [in] handle_t hBinding,
  [in] LONG SessionId,
  [in] LONG State,
  [in] ULONG Timeout
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which to wait for state change.

State: The desired state of the session, as specified in section 3.2.4.1.8, to wait for. The call will return when the session changes to this state.

Timeout: Maximum time, in milliseconds, to wait for the call to return.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.2.2 RpcRegisterAsyncNotification (Opnum 1)

The **RpcRegisterAsyncNotification** method registers for an event or events happening on a Terminal Server. The caller MUST call **RpcWaitAsyncNotification** after calling **RpcRegisterAsyncNotification** to wait for the notification.

```
HRESULT RpcRegisterAsyncNotification(
  [in] handle_t hBinding,
  [in] LONG SessionId,
  [in] TNotificationId Mask,
  [out] NOTIFY_HANDLE* phNotify
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which to wait for notification. If the value is -1, the call will register for notification for all sessions running on the Terminal Server.

Mask: Specifies the type of notification to wait for. This is of the type **TNotificationId**.

phNotify: Handle to the notification object. For more information, see **NOTIFY_HANDLE (section 2.2.1.21)**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.2.3 RpcWaitAsyncNotification (Opnum 2)

The **RpcWaitAsyncNotification** method starts the wait for the specified notification to be signaled by the Terminal Server. This MUST be called after **RpcRegisterAsyncNotification**. Because this is an asynchronous wait, it returns immediately.

```
HRESULT RpcWaitAsyncNotification(
  [in] NOTIFY_HANDLE hNotify,
  [out, size_is(, *pEntries)] PSESSION_CHANGE* SessionChange,
  [out] ULONG* pEntries
);
```

hNotify: The notification handle returned by **RpcRegisterAsyncNotification**.

SessionChange: An array of type **SESSION_CHANGE** containing information about all the sessions and the notifications received for that session.

pEntries: The number of entries returned in the *SessionChange* array.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.2.4 RpcUnRegisterAsyncNotification (Opnum 3)

The **RpcUnRegisterAsyncNotification** method cancels the asynchronous operation of waiting for notification from the Terminal Server. This MUST be called after **RpcRegisterAsyncNotification**.

```
HRESULT RpcUnRegisterAsyncNotification(
  [in, out] NOTIFY_HANDLE_EXCLUSIVE* phNotify
);
```

phNotify: The notification handle returned by **RpcRegisterAsyncNotification**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3 TermSrvEnumeration

The **TermSrvEnumeration** provides methods that provide for enumerating sessions and session information. The version for this interface is 1.0.

For information about endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcOpenEnum	Returns a handle of type **ENUM_HANDLE**, which can be used to query information about the sessions currently running on a Terminal Server. Opnum: 0
RpcCloseEnum	Closes the enumeration object returned by **RpcOpenEnum**. This method MUST be called after **RpcOpenEnum**. Opnum: 1
RpcFilterByState	Adds a filter to the session enumeration result, running on a Terminal Server, based on the state of the sessions. Opnum: 2
RpcFilterByCallersName	Adds a filter to the session enumeration result, running on a Terminal Server, based on the caller name. Opnum: 3
RpcEnumAddFilter	Adds another filter to the current enumeration. Opnum: 4
RpcGetEnumResult	Returns a structure of the type **PSESSIONENUM** containing the list of sessions currently running on the Terminal Server after applying the specified filter. Opnum: 5
RpcFilterBySessionType	Adds a filter to the session enumeration result, running on a Terminal Server, based on the type of the session. Opnum: 6
RpcFilterByLicenseType	Adds a filter to the session enumeration result, running on a Terminal Server, based on the type of license associated with the sessions. Opnum: 7
RpcGetSessionIds	Returns a list of the IDs associated with the sessions running on a Terminal Server that satisfies the specified filter. Opnum: 8

3.2.4.3.1 RpcOpenEnum (Opnum 0)

The **RpcOpenEnum** method returns a handle of the type **ENUM_HANDLE**, which can be used to query information about the sessions that are currently running on a Terminal Server.

```
HRESULT RpcOpenEnum(
  [in] handle_t hBinding,
  [out] ENUM_HANDLE* phEnum
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

phEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.2 RpcCloseEnum (Opnum 1)

The **RpcCloseEnum** method closes the enumeration object returned by **RpcOpenEnum**. This method MUST be called after **RpcOpenEnum**.

```
HRESULT RpcCloseEnum(
  [in, out] ENUM_HANDLE* phEnum
);
```

phEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.3 RpcFilterByState (Opnum 2)

The **RpcFilterByState** method adds a filter to the session enumeration result, running on a Terminal Server, based on the state of the sessions. This method MUST be called after **RpcOpenEnum** and before **RpcGetEnumResult**.

```
HRESULT RpcFilterByState(
  [in] ENUM_HANDLE hEnum,
  [in] LONG State,
  [in] BOOL bInvert
);
```

hEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

State: The session state, as specified in section 3.2.4.1.8, to be used to filter out enumeration result. Only the session with the specified state will be returned.

bInvert: Set to TRUE to imply that the result of the comparison during enumeration will be inverted, or FALSE otherwise.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.4 RpcFilterByCallersName (Opnum 3)

The **RpcFilterByCallersName** method adds a filter to the session enumeration result, running on a Terminal Server, based on the caller name. The enumeration will only return sessions belonging to the user making this call. This method MUST be called after **RpcOpenEnum** and before **RpcGetEnumResult**.

```
HRESULT RpcFilterByCallersName(
  [in] ENUM_HANDLE hEnum
);
```

hEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.5 RpcEnumAddFilter (Opnum 4)

The **RpcEnumAddFilter** method adds another filter to the current enumeration. This method MUST be called after **RpcOpenEnum**.

```
HRESULT RpcEnumAddFilter(
  [in] ENUM_HANDLE hEnum,
  [in] ENUM_HANDLE hSubEnum
);
```

hEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

hSubEnum: The handle to another enumeration whose filter will be applied to the current enumeration. The other enumeration MUST be created using **RpcOpenEnum** and any combination of **RpcFilterByState**, **RpcFilterByCallersName**, **RpcFilterBySessionType**, and **RpcFilterByLicenseType**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.6 RpcGetEnumResult (Opnum 5)

The **RpcGetEnumResult** method returns a structure of the type **PSESSIONENUM** containing the list of sessions currently running on the Terminal Server after applying the specified filter. This method MUST be called after **RpcOpenEnum**.

```
HRESULT RpcGetEnumResult(
  [in] ENUM_HANDLE hEnum,
  [out, size_is(,*pEntries)] PSESSIONENUM* ppSessionEnumResult,
  [in] DWORD Level,
  [out] ULONG* pEntries
);
```

hEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

ppSessionEnumResult: A structure of the type **PSESSIONENUM** containing the list of sessions currently running on the Terminal Server.

Level: The level of information requested. This field MUST be set to the only supported value of 3.

pEntries: The number of sessions currently running on the Terminal Server and returned in the *ppSessionEnumResult* structure.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.7 RpcFilterBySessionType (Opnum 6)

The **RpcFilterBySessionType** method adds a filter to the session enumeration result, running on a Terminal Server, based on the type of the session. This method MUST be called after **RpcOpenEnum** and before **RpcGetEnumResult**.

```
HRESULT RpcFilterBySessionType(
  [in] ENUM_HANDLE hEnum,
  [in] GUID* pSessionType
);
```

hEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

pSessionType: The session **GUID** to be used to filter out the enumeration result. Only the session with the specified **GUID** will be returned. **pSessionType** MUST be one of the possible **SessionType** values specified in section 2.2.2.4.1.2.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.8 RpcFilterByLicenseType (Opnum 7)

The **RpcFilterByLicenseType** method adds a filter to the session enumeration result, running on a Terminal Server, based on the type of license associated with the sessions. This method MUST be called after **RpcOpenEnum** and before **RpcGetEnumResult**.

```
HRESULT RpcFilterByLicenseType(
  [in] ENUM_HANDLE hEnum,
  [in] GUID* pLicenseType
);
```

hEnum: The handle to the session enumeration object. This is of type **ENUM_HANDLE**.

pLicenseType: The license type **GUID** to be used to filter out the enumeration result. Only the session associated with the specified license **GUID** will be returned. **pLicenseType** MUST be one of the possible **LicenseType** values specified in section 2.2.2.4.1.2.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.4.3.9 RpcGetSessionIds (Opnum 8)

The **RpcGetSessionIds** method returns a list of the IDs associated with the sessions running on a Terminal Server that satisfies the specified filter.

The server SHOULD enforce appropriate security measures to make sure the caller has required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, then the server MUST fail the call.

```
HRESULT RpcGetSessionIds(
  [in] handle_t hBinding,
  [in] SESSION_FILTER Filter,
  [in, range(0, 0xFFFF)] ULONG MaxEntries,
  [out, size_is(,*pcSessionIds)]
    LONG** pSessionIds,
  [out] ULONG* pcSessionIds
```

```
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

Filter: The filter to apply to the sessions running on the Terminal Server. This is of type **SESSION_FILTER**.

MaxEntries: The maximum number of session IDs to return. This value MUST be less than or equal to the number of entries in the *pSessionIds* array.

pSessionIds: An array to contain the list of session IDs running on the Terminal Server and filtered as specified by *Filter*.

pcSessionIds: The number of session IDs returned.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.2.5 Timer Events

No protocol timer events are required on the client except the timers that are required in the underlying **RPC transport**.

3.2.6 Other Local Events

No local events are used on the server except the events maintained in the underlying RPC transport.

3.3 TermSrv Client Details

3.3.1 Abstract Data Model

No abstract data model is required.

3.3.2 Timers

No protocol timers are required except those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE] section 3.3.3.2.1.

3.3.3 Initialization

The client MUST create an RPC connection to the Terminal Server using the details specified in section 2.1.

3.3.4 Message Processing Events and Sequencing Rules

This is a stateless protocol. No sequence of method calls is imposed on this protocol.

When a method completes, the values returned by RPC MUST be returned unmodified to the upper layer.

The client MUST ignore errors returned from the RPC server and notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client except the processing required in the underlying RPC protocol.

3.3.5 Timer Events

There are no timer events.

3.3.6 Other Local Events

There are no local events.

3.4 TermSrv Server Details

3.4.1 Abstract Data Model

No abstract data model is required.

3.4.2 Timers

No timers are required by this protocol.

3.4.3 Initialization

The parameters necessary to initialize the RPC protocol are specified in section 2.1.

3.4.4 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. See section 1.3 of [MS-RPCE] for more details.

All methods implemented by the TS server SHOULD enforce appropriate security measures to make sure the TS client has required permissions to execute the method. All methods MUST be RPC asynchronous calls.

The methods MAY throw an exception and the TS client MUST handle these exceptions gracefully.

3.4.4.1 RCMPublic

The **RCMPublic** interface provides methods that provide data about clients and sessions, and that enable shadowing sessions. The version for this interface is 1.0.

For information about endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcGetClientData	Returns information about the client that is connected to a particular session running on a Terminal Server. Opnum: 0

Method	Description
RpcGetConfigData	Returns the configuration data that is associated with the user connected to a particular session running on a Terminal Server. Opnum: 1
RpcGetProtocolStatus	Retrieves information about the status of the protocol that is used to connect to a particular session running on a Terminal Server. Opnum: 2
RpcGetLastInputTime	Returns the time the last user input was received for the specified session running on a Terminal Server by the associated protocol. Opnum: 3
RpcGetRemoteAddress	Retrieves the IP address of the client machine that is connected to the session on the Terminal Server. Opnum: 4
RpcShadow	Shadows a target session that is running on a Terminal Server from the session that initiates this call. Opnum: 5
RpcShadowTarget	Shadows a target session from a specified session that is running on a Terminal Server. Opnum: 6
RpcShadowStop	Stops all shadow operations on the specified session, including whether the session is acting as a shadow client or as a shadow target. Opnum: 7
RpcGetAllListeners	Returns a list of all Terminal Serviceslisteners running on a Terminal Server. Opnum: 8
RpcGetSessionProtocolLastInputTime	Returns the protocol status and time the last input was received by the protocol associated with a specific session running on a Terminal Server. Opnum: 9
RpcGetUserCertificates	Returns the client X509 certificate used to connect to a session running on a Terminal Server. Opnum: 10
RpcQuerySessionData	Returns information about a particular session running on a Terminal Server. Opnum: 11

3.4.4.1.1 RpcGetClientData (Opnum 0)

The **RpcGetClientData** method returns information about the client that is connected to a particular session running on a Terminal Server.

```
HRESULT RpcGetClientData(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [out, size_is(,*pOutBuffByteLen)]
    unsigned char** ppBuff,
  [out] ULONG* pOutBuffByteLen
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session from which client data is to be retrieved.

ppBuff: The buffer that contains the client data. This data is of type PWINSTATIONCLIENTW, specified in section 2.2.2.13.1. The buffer is allocated by **RpcGetClientData**.

pOutBuffByteLen: The number of bytes of client data that is returned.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.2 RpcGetConfigData (Opnum 1)

The **RpcGetConfigData** method returns the configuration data associated with the user connected to a particular session running on a Terminal Server.

```
HRESULT RpcGetConfigData(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [out, size_is(,*pOutBuffByteLen )]
    unsigned char** ppBuff,
  [out] ULONG* pOutBuffByteLen
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which the client configuration data is to be retrieved.

ppBuff: The buffer that will contain the client configuration data. This will be allocated by **RpcGetConfigData**. This data is of type PWINSTATIONCONFIG.

pOutBuffByteLen: The number of bytes of client configuration data that is returned.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.3 RpcGetProtocolStatus (Opnum 2)

The **RpcGetProtocolStatus** method retrieves information about the status of the protocol used to connect to a particular session running on a Terminal Server.

```
HRESULT RpcGetProtocolStatus(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [in] PROTOCOLSTATUS_INFO_TYPE InfoType,
  [out, size_is(,*pcbProtoStatus )]
    unsigned char** ppProtoStatus,
  [out] ULONG* pcbProtoStatus
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which protocol status is to be retrieved.

InfoType: Specifies what type of information to gather. This is of the type **PROTOCOLSTATUS_INFO_TYPE**.

ppProtoStatus: The buffer that will contain protocol status data. This data is of the type **PROTOCOLSTATUS**.

pcbProtoStatus: The number of bytes of protocol data that is returned.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific non-zero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.4 RpcGetLastInputTime (Opnum 3)

The **RpcGetLastInputTime** method returns the time the last user input was received for the specified sessions running on a Terminal Server by the associated protocol.

```
HRESULT RpcGetLastInputTime(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [out] hyper* pLastInputTime
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which the last user input time is to be retrieved.

pLastInputTime: The time when the last user input was received by the associated protocol.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.5 RpcGetRemoteAddress (Opnum 4)

The **RpcGetRemoteAddress** method retrieves the IP address of the client machine connected to the session on the Terminal Server.

```
HRESULT RpcGetRemoteAddress(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [out] PRCM_REMOTEADDRESS pRemoteAddress
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which client data is to be retrieved.

pRemoteAddress: The address of the client machine that is connected to the session. This is of the type **PRCM_REMOTEADDRESS**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.6 RpcShadow (Opnum 5)

The **RpcShadow** method shadows a target session, running on a Terminal Server, from the session that initiates this call.

```
HRESULT RpcShadow(
  [in] handle_t hBinding,
  [in, unique, string] WCHAR* szTargetServerName,
  [in] ULONG TargetSessionId,
  [in] BYTE HotKeyVk,
  [in] USHORT HotkeyModifiers
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

szTargetServerName: The name of the Terminal Services server on which the target session is running.

TargetSessionId: The ID of the session to shadow.

HotKeyVk: The virtual key code of the key to press to stop shadowing. This key is used in combination with the *HotkeyModifiers* parameter.

HotkeyModifiers: The virtual modifier that signifies the modifier key, such as SHIFT or CTRL, to depress to stop shadowing. The modifier key is used in combination with the key that is signified by the *HotKeyVk* parameter. This parameter can be any combination of KBDSHIFT, KBDCTRL, and KBDALT to indicate the SHIFT key, the CTRL key, and the ALT key respectively.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.7 RpcShadowTarget (Opnum 6)

The **RpcShadowTarget** method shadows a target session from a specified session running on a Terminal Server.

```
HRESULT RpcShadowTarget(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [in, size_is(ConfigSize)] PBYTE pConfig,
  [in, range(0, 0x8000)] ULONG ConfigSize,
  [in, size_is(AddressSize)] PBYTE pAddress,
  [in, range(0, 0x1000)] ULONG AddressSize,
  [in, size_is(ModuleDataSize)] PBYTE pModuleData,
  [in, range(0, 0x1000)] ULONG ModuleDataSize,
  [in, size_is(ThinwireDataSize)]
    PBYTE pThinwireData,
  [in, range(0, 0x1000)] ULONG ThinwireDataSize,
  [in, string] WCHAR* szClientName
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the target session to shadow.

pConfig: Pointer to WinStation configuration data. This parameter is of type **WINSTATIONCONFIG2**.

ConfigSize: The size, in bytes, of configuration data. Should be sizeof(WINSTATIONCONFIG2).

pAddress: Pointer to the address of the shadow client. This parameter is of type **CLIENT_STACK_ADDRESS**.

AddressSize: The size, in bytes, of the buffer that is pointed to by *pAddress*. Should be sizeof(**CLIENT_STACK_ADDRESS**).

pModuleData: Pointer to client module data previously retrieved from Terminal Server drivers. The contents of the data is TS driver specific. **RpcShadowTarget** sends this data as is to the TS drivers.

ModuleDataSize: The size, in bytes, of the client module data. This is at least MODULE_SIZE bytes.

pThinwireData: Pointer to the display driver data required for shadowing and previously retrieved from Terminal Server drivers. **RpcShadowTarget** sends this data as is to the TS drivers.

ThinwireDataSize: The size, in bytes, of the thinwire data. This is at least MODULE_SIZE bytes.

szClientName: Pointer to the client user name string (domain\username). This string must be NULL-terminated.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.8 RpcShadowStop (Opnum 7)

The **RpcShadowStop** method stops all shadow operations on the specified session, including whether the session is acting as a shadow client or as a shadow target.

```
HRESULT RpcShadowStop(
  [in] handle_t hBinding,
  [in] ULONG SessionId
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which shadow activity is to be stopped.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.9 RpcGetAllListeners (Opnum 8)

The **RpcGetAllListeners** method returns a list of all Terminal Services listeners running on a Terminal Server.

```
HRESULT RpcGetAllListeners(
  [in] handle_t hBinding,
  [out, size_is(,*pNumListeners)]
    PLISTENERENUM* ppListeners,
  [in] DWORD Level,
  [out] ULONG* pNumListeners
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

ppListeners: The list of Terminal Serviceslisteners running on the Terminal Server. This is an array of type **PLISTENERENUM**.

Level: The level of information that is requested for the listeners. The only supported value is 1.

pNumListeners: The number of listeners returned.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.10 RpcGetSessionProtocolLastInputTime (Opnum 9)

The **RpcGetSessionProtocolLastInputTime** method returns the protocol status and the time the last input was received by the protocol associated with a specific session running on a Terminal Server.

```
HRESULT RpcGetSessionProtocolLastInputTime(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [in] PROTOCOLSTATUS_INFO_TYPE InfoType,
  [out, size_is(,*pcbProtoStatus )]
    unsigned char** ppProtoStatus,
  [out] ULONG* pcbProtoStatus,
  [out] hyper* pLastInputTime
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session from which information is to be retrieved.

InfoType: Specifies what type of information to gather. This is of type **PROTOCOLSTATUS_INFO_TYPE**.

ppProtoStatus: The buffer that contains protocol status data. This data is of type PROTOCOLSTATUS, specified in section 2.2.2.14.3.

pcbProtoStatus: The number of bytes of protocol data returned.

pLastInputTime: The time the last input was received by the protocol.

Time is measured as the number of 100-nanosecond intervals since January 1, 1601 (UTC).

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000	Successful completion.

Return value/code	Description
S_OK	

3.4.4.1.11 RpcGetUserCertificates (Opnum 10)

The **RpcGetUserCertificates** method returns a client X509 certificate that is used to connect to a session running on a Terminal Server. For more information, see [X509].

```
HRESULT RpcGetUserCertificates(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [out] ULONG* pcCerts,
  [out, size_is(, *pcbCerts)] BYTE** ppbCerts,
  [out] ULONG* pcbCerts
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which the certificate is to be retrieved.

pcCerts: The number of client certificates returned.

ppbCerts: Certificate data.

pcbCerts: The size, in bytes, of *ppbCerts*.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.1.12 RpcQuerySessionData (Opnum 11)

The **RpcQuerySessionData** method returns information about a particular session running on a Terminal Server.

```
HRESULT RpcQuerySessionData(
  [in] handle_t hBinding,
  [in] ULONG SessionId,
  [in] QUERY_SESSION_DATA_TYPE type,
  [in, unique, size_is(cbInputData )]
    PBYTE pbInputData[],
  [in, range(0, 8192)] DWORD cbInputData,
  [out, ref, size_is(cbSessionData), length_is(*pcbReturnLength)]
    PBYTE pbSessionData[],
  [in, range(0, 8192)] ULONG cbSessionData,
  [out, ref] ULONG* pcbReturnLength,
  [out, ref] ULONG* pcbRequireBufferSize
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

SessionId: The ID of the session for which data is to be retrieved.

type: The type of data to retrieve about the session. This is of type **QUERY_SESSION_DATA_TYPE**.

pbInputData: Input Data. This is a string specifying the name of the virtual channel and is only required when querying virtual channel information.

cbInputData: The size, in bytes, of input data.

pbSessionData: The output data containing the requested information. The data returned is of type SESSIONDATA if type specified is **QUERY_SESSION_DATA_MODULE** and is of type WDCONFIG if type specified is **QUERY_SESSION_DATA_WDCONFIG**. Otherwise, it is protocol-specific.

cbSessionData: The size, in bytes, of **pbSessionData**.

pcbReturnLength: The length of the returned data, in bytes.

pcbRequireBufferSize: The buffer size required by the returned data.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.2 RCMListener

The **RCMListener** interface provides methods that open, close, start, and stop a listener. The version for this interface is 1.0.

For information about endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcOpenListener	Returns a handle to the specified Terminal Services listener running on a Terminal Server. Opnum: 0
RpcCloseListener	Closes the handle for a Terminal Services listener running on a Terminal Server. Opnum: 1
RpcStopListener	Stops the specified Terminal Services listener running on a Terminal Server. Opnum: 2
RpcStartListener	Starts the specified Terminal Services listener on a Terminal Server. Opnum: 3
RpcIsListening	Checks if the specified Terminal Services listener is running on a Terminal Server.

Method	Description
	Opnum: 4

3.4.4.2.1 RpcOpenListener (Opnum 0)

The **RpcOpenListener** method returns a handle to the specified Terminal Services listener running on a Terminal Server.

```
HRESULT RpcOpenListener(
  [in] handle_t hBinding,
  [in, string] WCHAR* szListenerName,
  [out] HLISTENER* phListener
);
```

hBinding: The RPC Binding Handle. For more information, see [MSDN-RPCBIND].

szListenerName: The name of the listener to retrieve a handle for.

phListener: Pointer to a handle to the listener. The handle is of type **HLISTENER**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.2.2 RpcCloseListener (Opnum 1)

The **RpcCloseListener** method closes the handle for a Terminal Services listener running on a Terminal Server. This MUST be called after **RpcOpenListener**.

```
HRESULT RpcCloseListener(
  [in, out] HLISTENER_EXCLUSIVE* phListener
);
```

phListener: Pointer to a handle to the listener as returned by **RpcOpenListener**. The handle is of type **HLISTENER**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.2.3 RpcStopListener (Opnum 2)

The **RpcStopListener** method stops the specified Terminal Services listener running on a Terminal Server. This MUST be called after **RpcOpenListener**.

```
HRESULT RpcStopListener(
  [in] HLISTENER hListener
);
```

hListener: The handle to the listener. This is of type **HLISTENER**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.2.4 RpcStartListener (Opnum 3)

The **RpcStartListener** method starts the specified Terminal Services listener on a Terminal Server. This MUST be called after **RpcOpenListener**.

```
HRESULT RpcStartListener(
  [in] HLISTENER hListener
);
```

hListener: The handle to the listener. This is of type **HLISTENER**.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.4.4.2.5 RpcIsListening (Opnum 4)

The **RpcIsListening** method checks if the specified Terminal Services listener is running on a Terminal Server. This MUST be called after **RpcOpenListener**.

```
HRESULT RpcIsListening(
  [in] HLISTENER hListener,
  [out] BOOL* pbIsListening
);
```

hListener: The handle to the listener. This is of type **HLISTENER**.

pbIsListening: Set to TRUE if the listener is listening for a connection, or FALSE otherwise.

Return Values: The method MUST return S_OK (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 S_OK	Successful completion.

3.5 Legacy Client Details

3.5.1 Abstract Data Model

No abstract data model is required.

3.5.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE].

3.5.3 Initialization

The client MUST create an RPC connection to the Terminal Server, using the details specified in section 2.1.

3.5.4 Message Processing Events and Sequencing Rules

This is a stateless protocol. No sequence of method calls is imposed on this protocol.

When a method completes, the values returned by RPC MUST be returned unmodified to the upper layer.

The client MUST ignore errors returned from the RPC server and notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.

3.5.5 Timer Events

There are no timer events.

3.5.6 Other Local Events

There are no local events.

3.6 Legacy Server Details

3.6.1 Abstract Data Model

No abstract data model is required.

3.6.2 Timers

No timers are required by this protocol.

3.6.3 Initialization

Parameters necessary to initialize the RPC protocol are specified in section 2.1.

3.6.4 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. For more information, see [MS-RPCE] section 1.3.

All methods implemented by the Terminal Services server SHOULD enforce appropriate security measures to make sure that the Terminal Services client has required permissions to execute the method. All methods MUST be RPC asynchronous calls.

The methods MAY throw an exception, and TS client MUST handle these exceptions gracefully.

Legacy server methods are part of the **LegacyApi** interface.

3.6.4.1 LegacyApi

The **LegacyApi** provides legacy methods that manipulate a Windows terminal client. The version for this interface is 1.0.

For endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcWinStationOpenServer	Returns a server handle that can be used in other WinStation API methods for querying information on the WinStation (sessions) on the server. Opnum: 0
RpcWinStationCloseServer	Closes the server handle for WinStation APIs. Opnum: 1
RpcIcaServerPing	Verifies that the server is alive. Opnum: 2
RpcWinStationEnumerate	Retrieves a list of **LOGONID** structures for sessions on a Terminal Server. Opnum: 3
RpcWinStationRename	Enables the caller to change the name of the session when it appears to utilities such as qwinsta and tsadmin. Opnum: 4
RpcWinStationQueryInformation	Retrieves various types of configuration information on a session. Opnum: 5
RpcWinStationSetInformation	Sets various types of configuration information for a session. Opnum: 6

Method	Description
RpcWinStationSendMessage	Displays a message box on a given Terminal Server session and, optionally, waits for a reply. Opnum: 7
RpcLogonIdFromWinStationName	Given a session name, returns the session's session ID. Opnum: 8
RpcWinStationNameFromLogonId	Retrieves the Windows Station (WinStation) name for a specific session. Opnum: 9
RpcWinStationConnect	Connects a user's Terminal Server client from a given Terminal Server session to a different Terminal Server session. Opnum: 10
RpcWinStationVirtualOpen	Opens a virtual channel to the Terminal Server. Opnum: 11
RpcWinStationBeepOpen	Opens a handle to the beep channel for a given session and process. Opnum: 12
RpcWinStationDisconnect	On the server, disconnects the Terminal Server client from a session. Opnum: 13
RpcWinStationReset	Resets a session. Opnum: 14
RpcWinStationShutdownSystem	Shuts down the system and, optionally, logs off all sessions. May also reboot the system. Opnum: 15
RpcWinStationWaitSystemEvent	Waits synchronously for a system event from an RPC API request on behalf of the caller. Opnum: 16
RpcWinStationShadow	Starts a shadow operation (remote control) of another Terminal Server session. Opnum: 17
RpcWinStationShadowTargetSetup	Internal RPC API called by the server on behalf of a shadower to set up a session shadow operation at the target end. Opnum: 18
RpcWinStationShadowTarget	Internal RPC API called by the server on behalf of a shadower to start a session shadow operation at the target end. Opnum: 19
Opnum20NotUsedOnWire	Not implemented.

Method	Description
	Opnum: 20
Opnum21NotUsedOnWire	Not implemented. Opnum: 21
Opnum22NotUsedOnWire	Not implemented. Opnum: 22
Opnum23NotUsedOnWire	Not implemented. Opnum: 23
Opnum24NotUsedOnWire	Not implemented. Opnum: 24
Opnum25NotUsedOnWire	Not implemented. Opnum: 25
RpcWinStationSetPoolCount	Sets the pool count for a given license. Opnum: 26
RpcWinStationQueryUpdateRequired	Queries the license on the server and determines if an update is required. Opnum: 27
RpcWinStationCallback	Given a phone number, calls back a client connected over a modem to the server. Opnum: 28
RpcWinStationBreakPoint	Breaks in to the debugger in either the csrss.exe process of a specific session or in the Terminal Server service. Opnum: 29
RpcWinStationReadRegistry	Tells the server to reread from the registry the configuration data for all the winstations. Opnum: 30
RpcWinStationWaitForConnect	Indicates the need to wait until there is a Terminal Serverclient connection to the session. Opnum: 31
RpcWinStationNotifyLogon	Notifies the server that a user has logged on to a session. Opnum: 32
RpcWinStationNotifyLogoff	Notifies the server that a user is in the process of logging off from a session. Opnum: 33
Opnum34NotUsedOnWire	Not implemented. Opnum: 34
RpcWinStationAnnoyancePopup	Displays a dialog on the session identified by LogonId to remind the user to register if necessary.

Method	Description
	MUST be called by winlogon processes when running as SYSTEM. Opnum: 35
RpcWinStationEnumerateProcesses	Returns the process information for an NT4 Terminal Server. Supported only for backward compatibility with that platform. Opnum: 36
RpcWinStationTerminateProcess	Terminates the specified process. Opnum: 37
Opnum38NotUsedOnWire	Reserved for local use. Opnum: 38
Opnum39NotUsedOnWire	Not implemented. Opnum: 39
Opnum40NotUsedOnWire	Not implemented. Opnum: 40
Opnum41NotUsedOnWire	Not implemented. Opnum: 41
RpcWinStationNtsdDebug	Sets up a connection for Ntsd to debug processes belonging to another session. Opnum: 42
RpcWinStationGetAllProcesses	Retrieves a list of the processes on a remote server on which the caller has permission to receive information. Opnum: 43
RpcWinStationGetProcessSid	Retrieves the process SID for a given process ID and process start time combination. Opnum: 44
RpcWinStationGetTermSrvCountersValue	Retrieves the current value of requested Terminal Server performance counters. Opnum: 45
RpcWinStationReInitializeSecurity	Reinitializes security for all non-console WinStation remote connection protocols specified in the registry. Opnum: 46
RpcWinStationBroadcastSystemMessage	Sends a message to a specified session running on the Terminal Server. Opnum: 47
RpcWinStationSendWindowMessage	Performs the equivalent of SendMessage to a window on a specific session. Opnum: 48
RpcWinStationNotifyNewSession	Called by winlogon to notify the Terminal Server

Method	Description
	service of a new session. Opnum: 49
Opnum50NotUsedOnWire	Not implemented. Opnum: 50
Opnum51NotUsedOnWire	Not implemented. Opnum: 51
Opnum52NotUsedOnWire	Not implemented. Opnum: 52
RpcWinStationGetLanAdapterName	Returns the LAN adapter **GUID** as a string. Opnum: 53
RpcWinStationUpdateUserConfig	Requests the server to update the user configuration data for a user logged in to a specific session. Opnum: 54
RpcWinStationQueryLogonCredentials	Checks autologon credentials for a session. Opnum: 55
RpcWinStationRegisterConsoleNotification	Registers a window in a console session to receive session notifications through a windows message. Opnum: 56
RpcWinStationUnRegisterConsoleNotification	Unregisters a window in a specific session to receive session notifications through a windows message. Opnum: 57
RpcWinStationUpdateSettings	Rereads settings for all WinStations. Opnum: 58
RpcWinStationShadowStop	Stops all shadow operations on the specified session. Opnum: 59
RpcWinStationCloseServerEx	Closes the server handle for WinStation APIs. Opnum: 60
RpcWinStationIsHelpAssistantSession	Determines whether a session is created by a HelpAssistant account (used for Remote Assistance). Opnum: 61
RpcWinStationGetMachinePolicy	Returns a copy of the Terminal Server machine policy to the caller. Opnum: 62
Opnum63NotUsedOnWire	Not implemented. Opnum: 63
Opnum64NotUsedOnWire	Not implemented. Opnum: 64

Method	Description
RpcWinStationCheckLoopBack	Checks if there is a loopback when a client tries to connect. Opnum: 65
RpcConnectCallback	Initiates a connection back to the Remote Assistance (RA) client. Opnum: 66
Opnum67NotUsedOnWire	Not implemented. Opnum: 67
Opnum68NotUsedOnWire	Not implemented. Opnum: 68
RpcRemoteAssistancePrepareSystemRestore	Prepares Terminal Server service for a Remote Assistance-specific system restore. Opnum: 69
RpcWinStationGetAllProcesses_NT6	Retrieves the processes running a remote server on which the caller has access to retrieve information. Opnum: 70
RpcWinStationRegisterNotificationEvent	Passes an event handle to be used when a notification should be sent. Opnum: 71
RpcWinStationUnRegisterNotificationEvent	Unregisters an event previously registered by **RpcWinStationRegisterNotificationEvent**. Opnum: 72
RpcWinStationAutoReconnect	Automatically reconnects a disconnected Terminal Server client to a given session. Opnum: 73
RpcWinStationCheckAccess	Determines if a user has specific permissions on a session. Opnum: 74
RpcWinStationOpenSessionDirectory	Pings the Session Directory to see if it can accept RPC calls. Opnum: 75

In the table above, the phrase "Reserved for local use" means the client MUST NOT send the **opnum**, and the server behavior is undefined<53> because it does not affect interoperability.

3.6.4.1.1 RpcWinStationOpenServer (Opnum 0)

The **RpcWinStationOpenServer** method returns a server handle that can be used in other WinStation API methods for querying information about the WinStation (sessions) on the server. In most cases, unless specified that they ignore the *hServer*, other WinStation API methods will return STATUS_UNSUCCESSFUL (as specified in [MS-ERREF]) in their own *pResult* parameter if the *hServer* parameter of that method is NULL.

The server SHOULD enforce appropriate security measures to make sure the caller has required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, then the server MUST fail the call.

```
BOOLEAN RpcWinStationOpenServer(
  [in] handle_t hBinding,
  [out] DWORD* pResult,
  [out] SERVER_HANDLE* phServer
);
```

hBinding: This parameter is not used.

pResult: Failure error code if the call to RpcWinStationOpenServer failed. If the call was successful, this parameter is STATUS_SUCCESS (0x00000000) (as specified in [MS-ERREF]).

phServer: Handle to the server object. This is of type **SERVER_HANDLE**. This handle is used by other RpcWinStation methods.

Return Values: Returns TRUE if the call succeeded, or FALSE if the lookup failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Lookup failed.

3.6.4.1.2 RpcWinStationCloseServer (Opnum 1)

The **RpcWinStationCloseServer** method closes the server handle for WinStation APIs.

```
BOOLEAN RpcWinStationCloseServer(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* MUST be returned from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationCloseServer failed. If the call was successful, this parameter is STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Return Values: Returns TRUE if the call succeeded, or FALSE if the call failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00	Method call failed.

Return value/code	Description
FALSE	

3.6.4.1.3 RpcIcaServerPing (Opnum 2)

The **RpcIcaServerPing** method is called to verify that the server is alive. <54>

```
BOOLEAN RpcIcaServerPing(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcIcaServerPing failed. If the call was successful, this parameter is STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.4 RpcWinStationEnumerate (Opnum 3)

The **RpcWinStationEnumerate** method retrieves a list of **LOGONID** structures for sessions on a Terminal Server.

```
BOOLEAN RpcWinStationEnumerate(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, out] PULONG pEntries,
  [in, out, unique, size_is(*pByteCount)]
    PCHAR pLogonId,
  [in, out] PULONG pByteCount,
  [in, out] PULONG pIndex
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The hServer argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationEnumerate** failed. If the call was successful, this parameter will be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

pEntries: Pointer to the number of entries to return to the caller. On return from this method, this is the number of logon IDs actually returned in this call to **RpcWinStationEnumerate**.

pLogonId: Buffer where the logon IDs are stored when the method returns. This will be an array of **LOGONID** structures. Caller should cast this to **PCHAR** before calling this method.

pByteCount: Size of the buffer pointed to by *pLogonId*.

pIndex: Last index of the logon ID lookup from this call. Should be passed to the server the next time this method is called. Initial value of this passed by the caller should be 0.

Return Values: Returns TRUE if the call succeeded, or FALSE if the lookup failed. On failure, *pResult* indicates the failure status code. If all of the logon IDs have already been retrieved from the server, TRUE will be returned, and pResult will be STATUS_NO_MORE_ENTRIES (as specified in [MS-ERREF]), indicating to the call that all logon IDs have been retrieved.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.5 RpcWinStationRename (Opnum 4)

The **RpcWinStationRename** method enables the caller to change the name of the session when it appears to utilities such as qwinsta and tsadmin. The caller must have DELETE permission on the session that is identified by the old name.<55>

```
BOOLEAN RpcWinStationRename(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, size_is(NameOldSize)] PWCHAR pWinStationNameOld,
  [in, range(0, 256)] DWORD NameOldSize,
  [in, size_is(NameNewSize)] PWCHAR pWinStationNameNew,
  [in, range(0, 256)] DWORD NameNewSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationRename** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Value	Meaning
STATUS_SUCCESS 0x00000000	Successful call.
STATUS_ACCESS_DENIED 0xC0000022	Caller does not have DELETE permission.
STATUS_CTX_WINSTATION_NAME_INVALID	The sizes are 0, one or the other of the pointers

Value	Meaning
0xC00A0001	is NULL, or a pointer is invalid.
STATUS_CTX_WINSTATION_NOT_FOUND 0xC00A0015	No session exists with the name given in *pWinStationNameOld*.
STATUS_CTX_WINSTATION_NAME_COLLISION 0xC00A0016	A session already exists with the name given in *pWinStationNameNew*.

pWinStationNameOld: Pointer to a string that is the old name of the session being renamed.

NameOldSize: Size of the old name in characters.

pWinStationNameNew: Pointer to a string that is the new name of the session being renamed.

NameNewSize: Size of the new name, in characters. Name MUST be shorter than or equal to WINSTATIONNAME_LENGTH.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.6 RpcWinStationQueryInformation (Opnum 5)

The **RpcWinStationQueryInformation** method retrieves various types of configuration information on a session. Multiple classes are allowed to retrieve different kinds of data. The caller MUST have WINSTATION_QUERY access, as well as specific access for some operations as indicated in the following sections.

```
BOOLEAN RpcWinStationQueryInformation(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] DWORD WinStationInformationClass,
  [in, out, unique, size_is(WinStationInformationLength)]
    PCHAR pWinStationInformation,
  [in, range(0, 0x8000)] DWORD WinStationInformationLength,
  [out] DWORD* pReturnLength
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationQueryInformation** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	Successful completion.
STATUS_INVALID_INFO_CLASS 0xC0000003	The class is not recognized.
STATUS_BUFFER_TOO_SMALL 0xC0000023	*WinStationInformationLength* is too small.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission for the operation.

LogonId: The session ID of the session for which to retrieve information.

WinStationInformationClass: The class of data to retrieve. These values come from the enum type **WINSTATIONINFOCLASS**.

The following classes are supported.

Value	Meaning
WinStationCreateData 0	Retrieves general information on the type of Terminal Server session (protocol) that the session belongs to, such as console or Rdp-Tcp. The *pWinStationInformation* argument points to a **WINSTATIONCREATEW** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONCREATEW**).
WinStationConfiguration 1	Retrieves general configuration data on the Terminal Server session. The *pWinStationInformation* argument points to a **WINSTATIONCONFIGW** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONCONFIGW**).
WinStationPdParams 2	Retrieves transport protocol driver parameters, such as data from tdtcp.sys in RDP implementations. The structure coming into the function indicates via SDClass the specific protocol driver to receive parameter information on. The result will be returned in the union in the structure. The *pWinStationInformation* points to a **PDPARAMSW** structure, and *WinStationInformationLength* should be sizeof(**PDPARAMSW**).
WinStationWd 3	Retrieves winstation protocol driver configuration data for the session, such as data from rdpwd.sys in RDP implementations. The *pWinStationInformation* argument points to a **WDCONFIGW** structure, and *WinStationInformationLength* should be sizeof(**WDCONFIGW**).
WinStationPd 4	Retrieves transport protocol driver configuration data for the session, such as data from tdtcp.sys for RDP implementations.

Value	Meaning
	The *pWinStationInformation* argument points to a **PDCONFIGW** structure, and *WinStationInformationLength* should be sizeof(**PDCONFIGW**).
WinStationPrinter 5	Not supported.
WinStationClient 6	Retrieves data on the Terminal Server client of the session. The *pWinStationInformation* argument points to a **WINSTATIONCLIENTW** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONCLIENTW**).
WinStationModules 7	Internal get function to retrieve data on protocol-specific binaries loaded for the given Terminal Server session. The structure pointed to by *pWinStationInformation* and the size of the buffer is Terminal Service protocol-specific.
WinStationInformation 8	Retrieves information on the session, including connect state, session's name, connect time, disconnect time, time last input was received from the client, logon time, user's username and domain, and the current time. pWinStationInformation points to a **WINSTATIONINFORMATIONW** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONINFORMATIONW**).
WinStationUserToken 14	Retrieves the user's token in the session. Caller needs WINSTATION_ALL_ACCESS permission. The *pWinStationInformation* argument points to a **WINSTATIONUSERTOKEN** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONUSERTOKEN**).
WinStationVideoData 16	Retrieves resolution and color depth of the session. The *pWinStationInformation* argument points to a **WINSTATIONVIDEODATA** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONVIDEODATA**).
WinStationCd 18	Retrieves connection driver configuration data. The *pWinStationInformation* points to a CDCONFIG structure, and *WinStationInformationLength* should be sizeof(CDCONFIG).
WinStationVirtualData 20	Internal get function to retrieve data on virtual channels for the given Terminal Server session. The structure pointed to by the *pWinStationInformation* argument and the size of the buffer is Terminal Service protocol-specific. However, size must be at least sizeof(**VIRTUALCHANNELNAME**).
WinStationLoadBalanceSessionTarget 24	Retrieves the target session ID for a client redirected from another server in a load balancing cluster. The *pWinStationInformation* points to a ULONG, and *WinStationInformationLength* should be sizeof(ULONG). If there is no redirection, -1 is returned in

Value	Meaning
	pWinStationInformation.
WinStationLoadIndicator 25	Retrieves an indicator of the load on the server. The *pWinStationInformation* argument points to a **WINSTATIONLOADINDICATORDATA** structure. *WinStationInformationLength* should be sizeof(**WINSTATIONLOADINDICATORDATA**).
WinStationShadowInfo 26	Retrieves the current shadow state of a session. The *pWinStationInformation* argument points to a **WINSTATIONSHADOW** structure. *WinStationInformationLength* should be sizeof(**WINSTATIONSHADOW**).
WinStationDigProductId 27	Retrieves the client product ID and current product ID of the session. The *pWinStationInformation* argument points to a **WINSTATIONPRODID** structure. *WinStationInformationLength* should be sizeof (**WINSTATIONPRODID**).
WinStationLockedState 28	Retrieves the current locked state of the session, TRUE or FALSE. The *pWinStationInformation* argument points to a BOOL variable. *WinStationInformationLength* should be sizeof (BOOL).
WinStationRemoteAddress 29	Retrieves the remote IP address of the Terminal Server client in the session. The *pWinStationInformation* argument points to a **WINSTATIONREMOTEADDRESS** structure. *WinStationInformationLength* should be sizeof (**WINSTATIONREMOTEADDRESS**).
WinStationIdleTime 30	Retrieves the idle time for the session, in seconds. The *pWinStationInformation* argument points to a ULONG variable. *WinStationInformationLength* should be sizeof (ULONG).
WinStationLastReconnectType 31	Retrieves the last reconnect type for the session. The value placed in *pWinStationInformation* will come from the enum **RECONNECT_TYPE**. The *pWinStationInformation* argument points to a ULONG variable. *WinStationInformationLength* should be sizeof (ULONG).
WinStationMprNotifyInfo 33	Retrieves the notification information for MPR (multiple protocol router) for this session. This MUST be called locally and by a process running as SYSTEM. The *pWinStationInformation* argument points to a **ExtendedClientCredentials** structure. *WinStationInformationLength* is sizeof(**ExtendedClientCredentials**).
WinStationExecSrvSystemPipe	Retrieves the name of the system pipe used by the

Value	Meaning
34	CreateRemoteProcess method that creates processes in different sessions than the caller. This is used internally by CreateRemoteProcess. Only processes running as SYSTEM can use the pipe name to connect to it. The *pWinStationInformation* argument points to a wide character string able to hold EXECSRVPIPENAMELEN characters, including the NULL terminator. *WinStationInformationLength* is EXECSRVPIPENAMELEN*sizeof(WCHAR).
WinStationSmartCardAutoLogon 35	Queries whether the session was a smartcard auto-logon, TRUE or FALSE, of the session. This MUST be called locally and by a process running as SYSTEM. The *pWinStationInformation* argument points to a variable of type BOOLEAN. *WinStationInformationLength* is sizeof(BOOLEAN).
WinStationIsAdminLoggedOn 36	Queries the server if the session has an Administrator logged on. This MUST be called locally and by a process running as SYSTEM. The *pWinStationInformation* argument points to a variable of type BOOLEAN. *WinStationInformationLength* is sizeof(BOOLEAN).
WinStationConnectState 37	Retrieves the connect state of the session. The *pWinStationInformation* argument points to a variable of type **WINSTATIONSTATECLASS**. *WinStationInformationLength* is sizeof(**WINSTATIONSTATECLASS**).

pWinStationInformation: Pointer to buffer allocated by the caller in which to retrieve the data. The data type or structure that *pWinStationInformation* points to is determined by the value of *WinStationInformationClass*. See previous sections for what *pWinStationInformation* should point to for each class.

WinStationInformationLength: Size of the data pointed to by *pWinStationInformation*, in bytes.

pReturnLength: Pointer to a variable to receive the size of the data retrieved. If *WinStationInformationLength* is too small, *pReturnLength* will indicate what the correct size should be, allocated by the caller.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.7 RpcWinStationSetInformation (Opnum 6)

The **RpcWinStationSetInformation** method sets various types of configuration information for a session. Multiple classes are allowed to set different kinds of data. The caller MUST have WINSTATION_SET access. Some operations MUST have more specific access as indicated in more detail in the sections that follow.<56>

```
BOOLEAN RpcWinStationSetInformation(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] DWORD WinStationInformationClass,
  [in, out, unique, size_is(WinStationInformationLength)]
    PCHAR pWinStationInformation,
  [in, range(0, 0x8000)] DWORD WinStationInformationLength
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationSetInformation** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_INVALID_INFO_CLASS 0xC0000003	The class is not recognized.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission for operation.

LogonId: The ID of the session to set information for.

WinStationInformationClass: The class of data to set. These values come from the enum type **WINSTATIONINFOCLASS**. See the following sections for the supported classes.

Value	Meaning
WinStationConfiguration 1	Merges configuration data into the Terminal Server session's data. The *pWinStationInformation* argument points to a **WINSTATIONCONFIGW** structure, and *WinStationInformationLength* should be sizeof(**WINSTATIONCONFIGW**).
WinStationPdParams 2	Sets transport protocol driver parameters. The structure coming into the function indicates via SDClass the specific protocol driver to set the parameter information for. The *pWinStationInformation* argument points to a **PDPARAMSW** structure, and *WinStationInformationLength* should be sizeof(**PDPARAMSW**).

Value	Meaning
WinStationTrace 9	Enables tracing on the lower level Terminal Server drivers for this session. This MUST be called by a process running as SYSTEM or as an Administrator. The *pWinStationInformation* argument points to a **TS_TRACE** structure, and *WinStationInformationLength* should be sizeof(**TS_TRACE**).
WinStationBeep 10	Sends a beep to the session. The *pWinStationInformation* argument points to a **BEEPINPUT** structure, and *WinStationInformationLength* should be sizeof(**BEEPINPUT**).
WinStationEncryptionOff 11	Turns encryption off.<57>
WinStationEncryptionPerm 12	Turns encryption permanently on.<58>
WinStationNtSecurity 13	Sends winlogon in the session a CTRL+ALT+DEL message. The *pWinStationInformation* argument and *WinStationInformationLength* are not used for this class. Set them to dummy valid data, however, as there always has to be something in those parameters.
WinStationInitialProgram 17	Tells winlogon that the process calling the server is the initial program to run instead of the explorer.exe file. This MUST be called for session IDs not equal to 0. The *pWinStationInformation* argument and *WinStationInformationLength* are not used for this class. Set them to dummy valid data, however, as there always has to be something in those parameters.
WinStationSystemTrace 19	Enables global tracing on the lower-level Terminal Server drivers. This MUST be called by a process running as SYSTEM or as an Administrator. The *pWinStationInformation* argument points to an **TS_TRACE** structure, and *WinStationInformationLength* should be sizeof(**TS_TRACE**).
WinStationClientData 21	Sends data to the Terminal Server client. *WinStationInformationLength* represents the complete length of all items to send, and must be at least sizeof(ULONG) + sizeof(**WINSTATIONCLIENTDATA**). Otherwise, STATUS_INFO_LENGTH_MISMATCH (as specified in [MS-ERREF]) is returned in pResult. If the data is larger than this but still less than what is expected, STATUS_INVALID_USER_BUFFER (as specified in [MS-ERREF]) is returned in *pResult*.
WinStationSecureDesktopEnter 22	Ignored by Terminal Server RDP drivers.
WinStationSecureDesktopExit 23	Ignored by Terminal Server RDP drivers.
WinStationShadowInfo	Sets the new shadow state of a session. The caller must be

Value	Meaning
26	local and running as SYSTEM. The *pWinStationInformation* argument points to a **WINSTATIONSHADOW** structure. *WinStationInformationLength* should be sizeof(**WINSTATIONSHADOW**).<59>
WinStationLockedState 28	Notifies processes of the new locked state of the session, TRUE or FALSE. The processes notified are those previously registered using either **RpcWinStationRegisterConsoleNotification** or **RpcWinStationRegisterNotificationEvent**. The *pWinStationInformation* argument points to a BOOL variable. *WinStationInformationLength* should be sizeof (BOOL).
WinStationDisallowAutoReconnect 32	Allows or disallows auto-reconnect behavior for this session, TRUE or FALSE. This MUST be called by a process running as SYSTEM. The *pWinStationInformation* argument points to a BOOLEAN value. *WinStationInformationLength* is sizeof(BOOLEAN).
WinStationMprNotifyInfo 33	Sets the notification information for multiple protocol router (MPR) for this session. This MUST be called locally and by a process running as SYSTEM. The *pWinStationInformation* argument points to a ExtendedClientCredentials structure. *WinStationInformationLength* is sizeof(ExtendedClientCredentials).
WinStationExecSrvSystemPipe 34	Sets the name of the system pipe used by the CreateRemoteProcess method that creates processes in different sessions than the caller. This is used internally by CreateRemoteProcess. The caller must be SYSTEM for this operation to succeed. The *pWinStationInformation* argument points to a wide character string able to hold EXECSRVPIPENAMELEN characters, including the NULL terminator. *WinStationInformationLength* is EXECSRVPIPENAMELEN*sizeof(WCHAR).

pWinStationInformation: Pointer to buffer allocated by the caller that the data for the operation is in. The data type or structure that *pWinStationInformation* points to is determined by the value of *WinStationInformationLength*.

WinStationInformationLength: Size of the data pointed to by *pWinStationInformation*, in bytes.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01	Successful completion.

Return value/code	Description
TRUE	
0x00 FALSE	Method call failed.

3.6.4.1.8 RpcWinStationSendMessage (Opnum 7)

The **RpcWinStationSendMessage** method displays a message box on a given Terminal Server session and, optionally, waits for a reply. The caller MUST have WINSTATION_MSG permission for this method to succeed.

```
BOOLEAN RpcWinStationSendMessage(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in, size_is(TitleLength)] PWCHAR pTitle,
  [in, range(0, 1024 )] DWORD TitleLength,
  [in, size_is(MessageLength)] PWCHAR pMessage,
  [in, range(0, 1024 )] DWORD MessageLength,
  [in] DWORD Style,
  [in] DWORD Timeout,
  [out] DWORD* pRespsonse,
  [in] BOOLEAN DoNotWait
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationSendMessage** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have WINSTATION_MSG permission.

LogonId: The session ID of the session to display the message box on.

pTitle: Pointer to the title for the message box to display.

TitleLength: The length, in bytes, of the title to display.

pMessage: Pointer to the message to display.

MessageLength: The length, in bytes, of the message to display in the specified session.

Style: Can be any value that the standard Windows MessageBox() method's Style parameter can take. For more information, see [MSDN-MSGBOX].

Timeout: The response timeout, in seconds. If the message box is not responded to in *Timeout* seconds, a response code of IDTIMEOUT MUST be returned in *pRespsonse* to indicate that the message box timed out.

pRespsonse: The return code from the MessageBox method. This value will be standard MessageBox return values. For more information, see [MSDN-MSGBOX].

DoNotWait: If set to TRUE, do not wait for the response to the message. If set to FALSE, on return the *pRespsonse* parameter will be set to IDASYNC if no errors occur in queuing the message.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.9 RpcLogonIdFromWinStationName (Opnum 8)

The **RpcLogonIdFromWinStationName** method returns a session's session ID given its session name. The caller must have WINSTATION_QUERY permission.

```
BOOLEAN RpcLogonIdFromWinStationName(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, size_is(NameSize)] PWCHAR pWinStationName,
  [in, range(0, 256 )] DWORD NameSize,
  [out] DWORD* pLogonId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcLogonIdFromWinStationName** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission for operation.

pWinStationName: Pointer to a buffer holding the session name.

NameSize: The size of the string pointed to by pWinStationName, in characters. MUST be less than or equal to WINSTATIONNAME_LENGTH.

pLogonId: The matching session ID for the session specified by *pWinStationName*.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.10 RpcWinStationNameFromLogonId (Opnum 9)

The **RpcWinStationNameFromLogonId** method retrieves the Windows Station (WinStation) name for a specific session. Examples of the format of the name include console, rdp-tcp, rdp-tcp#3 for the console session, RDP TCP listener session or third session created from the RDP TCP listener. The caller MUST have WINSTATION_QUERY permission.

```
BOOLEAN RpcWinStationNameFromLogonId(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LoginId,
  [in, out, size_is(NameSize)] PWCHAR pWinStationName,
  [in, range(0, 256 )] DWORD NameSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationNameFromLogonId** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

LoginId: The ID of the session to retrieve the WinStation name for.

pWinStationName: Pointer to a buffer in which to store the WinStation name.

NameSize: The size, in bytes, of the buffer where the WinStation name will be stored. MUST be less than or equal to WINSTATIONNAME_LENGTH.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.11 RpcWinStationConnect (Opnum 10)

The **RpcWinStationConnect** method connects a user's Terminal Server client from a given Terminal Server session to a different Terminal Server session. If there is a client connected to the target session, that client will be disconnected.

The client indicated by *ConnectLogonId* must have WINSTATION_CONNECT and WINSTATION_DISCONNECT access. Similarly *TargetLogonId* must have WINSTATION_DISCONNECT access.

```
BOOLEAN RpcWinStationConnect(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId,
  [in] DWORD ConnectLogonId,
  [in] DWORD TargetLogonId,
  [in, size_is(PasswordSize)] PWCHAR pPassword,
  [in, range(0, 1024)] DWORD PasswordSize,
  [in] BOOLEAN Wait
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationConnect** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

ClientLogonId: The session ID of the caller of this method.

ConnectLogonId: The ID of the session from which the connection is being made. This MUST be the same as *TargetLogonId* and MUST be an existing session ID. The user MUST be logged on. To indicate the current session, this MUST be LOGONID_CURRENT.

TargetLogonId: The session ID of the session to which the connection is being made. Cannot be the same as *ConnectLogonId* and MUST be an existing session ID.

pPassword: The password of *LogonId*'s session. This SHOULD be NULL if the same user is making the call. The password MUST be valid or NULL, or else an error will be returned.

PasswordSize: The size of *pPassword*.

Wait: TRUE indicates to wait for the connect to complete, FALSE otherwise.<60>

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.12 RpcWinStationVirtualOpen (Opnum 11)

The **RpcWinStationVirtualOpen** method opens a virtual channel to the Terminal Server. This call MUST be made by processes local on the server. The caller must have WINSTATION_VIRTUAL permission and PROCESS_DUP_HANDLE (0x0040) permission on Pid.<61>

```
BOOLEAN RpcWinStationVirtualOpen(
```

```
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] DWORD Pid,
  [in, size_is(NameSize)] PCHAR pVirtualName,
  [in, range(0, 1024)] DWORD NameSize,
  [out] ULONG_PTR* pHandle
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationVirtualOpen** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission for operation.

LogonId: The ID of the session to open the virtual channel on. MUST be a valid session ID.

Pid: The ID of the process for which to open the virtual channel.

pVirtualName: The name of the virtual channel to open.

NameSize: The length of *pVirtualName*. MUST be less than or equal to VIRTUALCHANNELNAME_LENGTH + 1 in length and cannot be zero. Otherwise, STATUS_INVALID_PARAMETER (0xC000000D), as specified in [MS-ERREF], will be returned in *pResult*.

pHandle: Pointer to the handle for the virtual session.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.13 RpcWinStationBeepOpen (Opnum 12)

The **RpcWinStationBeepOpen** method opens a handle to the beep channel for a given session and process. The caller MUST have WINSTATION_VIRTUAL access and the *LogonId* MUST be the same as the caller unless the process is running as SYSTEM. On these platforms, the method is defined as follows:

```
BOOLEAN RpcWinStationBeepOpen(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] DWORD Pid,
  [out] ULONG_PTR* pHandle
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationAnnoyancePopup failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	Returned if the caller is in a different session than *LogonId* and caller is not running as SYSTEM. Or, returned if the caller does not have WINSTATION_VIRTUAL permission.

LogonId: A session where the beep is to be played.

Pid: The process that will play the beep.

pHandle: Pointer to a variable to hold a handle to the beep channel. The handle can be treated like a normal beep device. When the caller is done with this handle, the caller should call CloseHandle on the handle.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.14 RpcWinStationDisconnect (Opnum 13)

The **RpcWinStationDisconnect** method disconnects, on the server, the Terminal Server client from a session. The caller of this method must have WINSTATION_DISCONNECT permission on the session to disconnect.

```
BOOLEAN RpcWinStationDisconnect(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] BOOLEAN bWait
```

```
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationDisconnect failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission to disconnect the session.

LogonId: The ID of the session to disconnect. Can be LOGONID_CURRENT to indicate the current session.

bWait: TRUE to wait for the disconnect to complete before returning, FALSE otherwise.<62>

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.15 RpcWinStationReset (Opnum 14)

The **RpcWinStationReset** method resets a session. Resetting a session will lead to the user being logged off and his or her Terminal Server client being disconnected. In the case of a remote non-console Terminal Server session or a session other than the primary session in a Fast User Switching (FUS) scenario, the session and its processes will also terminate. The caller MUST have WINSTATION_RESET permissions.<63>

```
BOOLEAN RpcWinStationReset(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] BOOLEAN bWait
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationReset failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have WINSTATION_RESET permission.

LogonId: The ID of the session to reset.

bWait: TRUE to wait for the disconnect to complete before returning, FALSE otherwise.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.16 RpcWinStationShutdownSystem (Opnum 15)

The **RpcWinStationShutdownSystem** method shuts down the system and optionally logs off all sessions and/or reboots the system. The caller needs SE_SHUTDOWN_PRIVILEGE [WININTERNALS] when performing the shutdown locally and SE_REMOTE_SHUTDOWN_PRIVILEGE [WININTERNALS] when performing the shutdown remotely. The caller calls ExitWindowsEx [WININTERNALS] to perform the actual shutdown once all checks have been completed.

```
BOOLEAN RpcWinStationShutdownSystem(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId,
  [in] DWORD ShutdownFlags
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationShutdownSystem failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission to shutdown the system.

ClientLogonId: The session requesting to shutdown the system. Ignored when the RPC call is remote.

ShutdownFlags: Shutdown flags. Can be any combination of the following flags.

Value	Meaning
WSD_LOGOFF 0x00000001	Forces sessions to logoff.
WSD_SHUTDOWN 0x00000002	Shuts down the system.
WSD_REBOOT 0x00000004	Reboots after shutdown.
WSD_POWEROFF 0x00000008	Powers off after shutdown.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.17 RpcWinStationWaitSystemEvent (Opnum 16)

The **RpcWinStationWaitSystemEvent** method waits synchronously for a system event from an RPC API request on behalf of the caller. There is no timeout on the wait. Only one event wait may be posted per server handle at a time. The code protects itself from misuse by returning STATUS_PIPE_BUSY (0xC00000AE), as specified in [MS-ERREF], if an event wait is already outstanding, and the request is not a cancel. The first time this is called the server will create an event block for the handle specified by hServer. This event block will be cleared if **RpcWinStationWaitSystemEvent** is called with *EventMask* equal to WEVENT_NONE or if **RpcWinStationCloseServer** or **RpcWinStationCloseServerEx** are closed for the handle *hServer*.

```
BOOLEAN RpcWinStationWaitSystemEvent(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD EventMask,
  [out] DWORD* pEventFlags
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationWaitSystemEvent failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

EventMask: The mask of events to wait for. Can be any combination of the following except for WEVENT_NONE:

Value	Meaning
WEVENT_NONE 0x00000000	The client desires to clear its event wait block. SHOULD be called when completing waiting for event. When RpcWinStationCloseServer is called for hServer, this is called on the client's behalf.
WEVENT_CREATE 0x00000001	Wait for a new session to be created.
WEVENT_DELETE 0x00000002	Wait for an existing session to be deleted.
WEVENT_RENAME 0x00000004	Wait for a session to be renamed.
WEVENT_CONNECT 0x00000008	The session connected to a client.
WEVENT_DISCONNECT 0x00000010	A client disconnected from the session.
WEVENT_LOGON 0x00000020	A user logged on to session.
WEVENT_LOGOFF 0x00000040	A user logged off from session.
WEVENT_STATECHANGE 0x00000080	The session state changed.
WEVENT_LICENSE 0x00000100	The license state changed. This is not used in RDP.
WEVENT_ALL 0x7fffffff	Wait for all event types.
WEVENT_FLUSH 0x80000000	Release all waiting clients.

pEventFlags: Pointer to a variable to receive a bitmask that is a subset of *EventMask* indicating which events actually occurred during this wait operation.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.18 RpcWinStationShadow (Opnum 17)

The **RpcWinStationShadow** method starts a shadow (remote control) operation of another Terminal Server session. The other session can be local or on a Terminal Server. The session to shadow MUST be in the active state with a user logged on.<64>

```
BOOLEAN RpcWinStationShadow(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in, unique, size_is(NameSize)]
    PWCHAR pTargetServerName,
  [in, range(0, 1024)] DWORD NameSize,
  [in] DWORD TargetLogonId,
  [in] BYTE HotKeyVk,
  [in] USHORT HotkeyModifiers
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument value MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationShadow failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

LogonId: The ID of the session to shadow from.

pTargetServerName: The shadow target server name. MAY be NULL to indicate the current server.

NameSize: The length of *pTargetServerName* in bytes. MAY be 0 if *pTargetServerName* is NULL.

TargetLogonId: The shadow target session ID.

HotKeyVk: The virtual key code of the key to press to stop shadowing. This key is used in combination with the *HotkeyModifiers* parameter.

HotkeyModifiers: The virtual modifier that signifies the modifier key, such as shift or control, to depress to stop shadowing. The modifier key is used in combination with the key signified by the *HotKeyVk* parameter. This parameter MAY be any combination of KBDSHIFT, KBDCTRL, and KBDALT to indicate the SHIFT key, the CTRL key and the ALT key, respectively.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.19 RpcWinStationShadowTargetSetup (Opnum 18)

The **RpcWinStationShadowTargetSetup** method is an internal RPC API called by the server on behalf of a shadower to set up a session shadow operation at the target end. This method should be called before **RpcWinStationShadowTarget**. When **RpcWinStationShadow** is called with a remote server name, the server MUST call this API on the remote server first prior to calling **RpcWinStationShadowTarget** at the remote server.<65>

```
BOOLEAN RpcWinStationShadowTargetSetup(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationShadowTargetSetup** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

LogonId: The ID of the session to shadow.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.<66>

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.20 RpcWinStationShadowTarget (Opnum 19)

The **RpcWinStationShadowTarget** method is an internal RPC API called by the server on behalf of a shadower to start a session shadow operation at the target end. When **RpcWinStationShadow** is called with a remote server name, the server will call this API on the remote server after first calling **RpcWinStationShadowTargetSetup** at the remote server.

```
BOOLEAN RpcWinStationShadowTarget(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in, size_is(ConfigSize)] PBYTE pConfig,
  [in, range(0, 0x8000)] DWORD ConfigSize,
  [in, size_is(AddressSize)] PBYTE pAddress,
  [in, range(0, 0x1000 )] DWORD AddressSize,
  [in, size_is(ModuleDataSize)] PBYTE pModuleData,
  [in, range(0, 0x1000 )] DWORD ModuleDataSize,
  [in, size_is(ThinwireDataSize)]
    PBYTE pThinwireData,
  [in] DWORD ThinwireDataSize,
  [in, size_is(ClientNameSize)] PBYTE pClientName,
  [in, range(0, 1024 )] DWORD ClientNameSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationShadowTarget** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

LogonId: The ID of the session to shadow.

pConfig: Pointer to WinStation config data. This parameter is of type **WINSTATIONCONFIG2**.

ConfigSize: The size of config data, in bytes. Should be sizeof(**WINSTATIONCONFIG2**).

pAddress: Pointer to the address of the shadow client. This parameter is of type **CLIENT_STACK_ADDRESS**.

AddressSize: The size of the buffer pointed to by pAddress, in bytes. Should be sizeof(**CLIENT_STACK_ADDRESS**).

pModuleData: Pointer to client module data previously retrieved from Terminal Server drivers. The contents of the data is TS driver-specific. **RpcWinStationShadowTarget** sends this data as is to the Terminal Server drivers.

ModuleDataSize: The size of the client module data, in bytes. This is at least MODULE_SIZE bytes.

pThinwireData: Pointer to the thinwire module data previously retrieved from Terminal Server drivers. This is display driver data required for shadowing and is Terminal Server driver-specific. **RpcWinStationShadowTarget** sends this data as is to the TS drivers.

ThinwireDataSize: The size of the thinwire data, in bytes. This is at least MODULE_SIZE bytes.

pClientName: Pointer to client user name string (domain\username). This string MUST be NULL-terminated.

ClientNameSize: The size of the client name string.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.21 RpcWinStationSetPoolCount (Opnum 26)

The **RpcWinStationSetPoolCount** method sets the pool count for a given license. This is not available with RDP.<67>

```
BOOLEAN RpcWinStationSetPoolCount(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, out, size_is(LicenseSize)]
    PCHAR pLicense,
  [in, range(0, 0x4000)] DWORD LicenseSize
```

```
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationSetPoolCount** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_UNSUCCESSFUL 0xC0000001	Always returned when there are only TS RDP sessions.

pLicense: Pointer to the license to set the pool count for.

LicenseSize: The size of the license, in bytes.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.22 RpcWinStationQueryUpdateRequired (Opnum 27)

The **RpcWinStationQueryUpdateRequired** method queries the license on the server and determines if an update is required.<68>

```
BOOLEAN RpcWinStationQueryUpdateRequired(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [out] DWORD* pUpdateFlag
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationQueryUpdateRequired** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.

Value	Meaning
STATUS_UNSUCCESSFUL 0xC0000001	The Terminal Server protocol extension DLL does not support this operation or RDP does not support this.

pUpdateFlag: Set to TRUE if an update is required, or FALSE otherwise.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.23 RpcWinStationCallback (Opnum 28)

Given a phone number, the **RpcWinStationCallback** method calls back a client connected over a modem to the server. This method MUST be called by a process running as SYSTEM. This is not available with RDP.<69>

```
BOOLEAN RpcWinStationCallback(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in, size_is(PhoneNumberSize)]
    PWCHAR pPhoneNumber,
  [in, range(0, 0x1000 )] DWORD PhoneNumberSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationCallback** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have SYSTEM privileges.

LogonId: The ID of the session where the client will be disconnected, then called back.

pPhoneNumber: Pointer to the string of the client phone number to call back.

PhoneNumberSize: The size of the phone number string pointed to by *pPhoneNumber*, in bytes, including the NULL terminator character.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.24 RpcWinStationBreakPoint (Opnum 29)

The **RpcWinStationBreakPoint** method breaks into the debugger in either the csrss.exe process of a specific session or in the Terminal Server service. This method MUST be called by Administrators because the SE_SHUTDOWN_PRIVILEGE [WININTERNALS] privilege is enabled.<70>

```
BOOLEAN RpcWinStationBreakPoint(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] BOOLEAN KernelFlag
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationBreakPoint** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The SE_SHUTDOWN_PRIVILEGE [WININTERNALS] privilege cannot be enabled.

LogonId: The ID of the session to break into the debugger in the csrss.exe process. If this parameter is -2, the Terminal Server service SHOULD break into the debugger instead.<71>

KernelFlag: Set to TRUE to indicate that the server will break into the debugger in a particular session in kernel mode. If *LogonId* is -2, the server MUST break into the debugger in user mode.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00	Method call failed.

Return value/code	Description
FALSE	

3.6.4.1.25 RpcWinStationReadRegistry (Opnum 30)

The **RpcWinStationReadRegistry** method tells the server to reread, from the registry, the configuration data for all the winstations. The registry reread is HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server\WinStations for each subkey under WinStation. The caller of this RPC method MUST be running as either System or as an Administrator.<72>

```
BOOLEAN RpcWinStationReadRegistry(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationReadRegistry** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.26 RpcWinStationWaitForConnect (Opnum 31)

The **RpcWinStationWaitForConnect** method indicates to wait until there is a Terminal Server client connection to the session indicated by *ClientLogonId*. This method MUST be called by winlogon process running as SYSTEM and MUST be called before someone has connected to the session.<73>

```
BOOLEAN RpcWinStationWaitForConnect(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId,
  [in] DWORD ClientProcessId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationWaitForConnect** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not running as SYSTEM SID and is not the winlogon process for the session. This value is also returned if the session has already had a client connect to it at least once.

ClientLogonId: The ID of the session to wait until a Terminal Server client connects.

ClientProcessId: The ID of the process requesting to wait until a Terminal Server client connects. This MUST be the same process ID of the winlogon.exe process in the session.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.27 RpcWinStationNotifyLogon (Opnum 32)

The **RpcWinStationNotifyLogon** method notifies the server that a user has logged on to a session. This API MUST be called locally by threads in winlogon processes running as SYSTEM.<74>

```
BOOLEAN RpcWinStationNotifyLogon(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId,
  [in] DWORD ClientProcessId,
  [in] BOOLEAN fUserIsAdmin,
  [in] DWORD UserToken,
  [in, size_is(DomainSize)] PWCHAR pDomain,
  [in, range(0, 1024 )] DWORD DomainSize,
  [in, size_is(UserNameSize)] PWCHAR pUserName,
  [in, range(0, 1024 )] DWORD UserNameSize,
  [in, size_is(PasswordSize)] PWCHAR pPassword,
  [in, range(0, 1024 )] DWORD PasswordSize,
  [in] UCHAR Seed,
  [in, size_is(ConfigSize)] PCHAR pUserConfig,
  [in, range(0, 0x8000 )] DWORD ConfigSize,
  [out] BOOLEAN* pfIsRedirected
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationNotifyLogon** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not SYSTEM, is not winlogon, or is not in the same session as specified by the *ClientLogonId* parameter.
STATUS_ACCESS_VIOLATION 0xC0000005	*ConfigSize* is less than sizeof(**USERCONFIGW**).

ClientLogonId: The session ID that logged on a user.

ClientProcessId: The process ID making this request. MUST be the winlogon process in the session specified by the *ClientLogonId* parameter.

fUserIsAdmin: Set to TRUE if the user is an administrator, or FALSE if the user is not an administrator.

UserToken: The user's token.

pDomain: Pointer to a buffer containing the user's domain name. MUST be NULL-terminated.

DomainSize: The size of the user's domain name, in bytes. MUST be less than or equal to DOMAIN_LENGTH.

pUserName: Pointer to a buffer where the user's user name is provided. MUST be NULL-terminated.

UserNameSize: The size of the user's user name, in bytes. MUST be less than or equal to USERNAME_LENGTH.

pPassword: Pointer to a buffer containing the user's password. MUST be NULL-terminated.

PasswordSize: the size of the user's password, in bytes.

Seed: Ignored. SHOULD be 0.

pUserConfig: Pointer to a buffer containing the user configuration data. This buffer MUST be of type **USERCONFIGW**.

ConfigSize: Size of the buffer pointed to by pUserConfig in bytes. MUST be equal to or greater than sizeof(**USERCONFIGW**).

pfIsRedirected: Pointer to a variable to receive a Boolean value indicating whether the Session Directory requested this user be redirected to a different server. If TRUE is returned in this parameter, STATUS_UNSUCCESSFUL, as specified in [MS-ERREF], MUST be returned in pResult.<75>

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.28 RpcWinStationNotifyLogoff (Opnum 33)

The **RpcWinStationNotifyLogoff** method notifies the server that a user is in the process of logging off from a session. This API MUST be called locally by threads in winlogon processes running as SYSTEM. **RpcWinStationNotifyLogon** should be called before this function.<76>

```
BOOLEAN RpcWinStationNotifyLogoff(
  [in] SERVER_HANDLE hServer,
  [in] DWORD ClientLogonId,
  [in] DWORD ClientProcessId,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

ClientLogonId: The session ID that logged off.

ClientProcessId: The process ID that is making this request. MUST be the winlogon process in the session specified by the *ClientLogonId* parameter.

pResult: Failure error code if the call to **RpcWinStationNotifyLogoff** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not SYSTEM, is not winlogon, or is not in the same session as specified by the *ClientLogonId* parameter.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.29 RpcWinStationAnnoyancePopup (Opnum 35)

The **RpcWinStationAnnoyancePopup** method MUST be called by winlogon processes when running as SYSTEM. Displays a dialog on the session identified by *LogonId* to remind the user to register if necessary.<77> <78>

```
BOOLEAN RpcWinStationAnnoyancePopup(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonIdld
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationAnnoyancePopup failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not winlogon or is not running as SYSTEM.
STATUS_UNSUCCESSFUL 0xC0000001	WsxWinStationLogonAnnoyance [WININTERNALS] is not implemented in the protocol DLL used for the session identified by *LogonId*.

LogonIdld: The session ID to display the dialog on.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.<79>

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.30 RpcWinStationEnumerateProcesses (Opnum 36)

The **RpcWinStationEnumerateProcesses** method retrieves the processes running on a remote server that the caller has access to retrieve information on.<80>

```
BOOLEAN RpcWinStationEnumerateProcesses(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [out, size_is(ByteCount)] PBYTE pProcessBuffer,
  [in, range(0, 0x8000 )] DWORD ByteCount
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to RpcWinStationEnumerateProcesses failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	*ByteCount* is too small to receive the data.

pProcessBuffer: Pointer to a buffer receiving the list of processes.

In *pProcessBuffer* for each process, the server will return the following in the buffer, in this order:

1. A **_TS_SYS_PROCESS_INFORMATION_NT4** structure.
2. A number of SYSTEM_THREAD_INFORMATION [WININTERNALS] structures equal to the maximum number of threads in the process.
3. A **TS_PROCESS_INFORMATION** for the process.
4. The image name for the process pointed to by the ImageName field in the **_TS_SYS_PROCESS_INFORMATION_NT4** structure.

Only up to SIZEOF_TS4_SYSTEM_THREAD_INFORMATION of the SYSTEM_THREAD_INFORMATION structure (as specified in [WININTERNALS]) is copied into the buffer and only up to SIZEOF_TS4_SYSTEM_PROCESS_INFORMATION of the **_TS_SYS_PROCESS_INFORMATION_NT4** structure is copied into the buffer.

ByteCount: The size, in bytes, of the *pProcessBuffer* parameter. If ByteCount is too small to receive the data, the method returns an error code (STATUS_INFO_LENGTH_MISMATCH, as specified in [MS-ERREF]) in the *pResult* parameter. Note that there is no indication during failure given to the caller as to what the correct size should be if *pProcessBuffer* is too small.

Return Values: Returns TRUE if the call succeeded, or FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.31 RpcWinStationTerminateProcess (Opnum 37)

The **RpcWinStationTerminateProcess** method terminates the specified process. An attempt is made to enable the SE_DEBUG_PRIVILEGE privilege to kill processes not owned by the current user,

including processes running in other Terminal Server sessions. Caller MUST have terminate access to terminate the process.

```
BOOLEAN RpcWinStationTerminateProcess(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ProcessId,
  [in] DWORD ExitCode
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. hServer MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationTerminateProcess** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000), as specified in [MS-ERREF].

Value	Meaning
STATUS_SUCCESS 0x00000000	The call succeeded.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission to terminate the process.

ProcessId: The ID of the process to terminate.

ExitCode: The termination status for each thread in the process.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.32 RpcWinStationNtsdDebug (Opnum 42)

The RpcWinStationNtsdDebug method sets up a connection for Ntsd to debug processes belonging to another session. MUST be called locally by processes having WINSTATION_ALL_ACCESS permission. This method MUST fail with error code STATUS_SERVER_DISABLED unless cross session debugging is enabled under the registry key \\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server.<81>

```
BOOLEAN RpcWinStationNtsdDebug(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] LONG ProcessId,
  [in] ULONG DbgProcessId,
```

```
  [in] ULONG DbgThreadId,
  [in] DWORD_PTR AttachCompletionRoutine
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationNtsdDebug** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

LogonId: The session to debug. This MUST be a valid session ID or STATUS_CTX_WINSTATION_NOT_FOUND will be returned in pResult.

ProcessId: The ID of the process to debug.

DbgProcessId: The ID of the debugger process.

DbgThreadId: The thread ID of the debugger process.

AttachCompletionRoutine: The routine called once the attach from *DbgProcessId* to *ProcessId* has completed. A remote thread is created within the process being debugged that will call this method once the debug has been set up.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, *pResult* indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.33 RpcWinStationGetAllProcesses (Opnum 43)

The **RpcWinStationGetAllProcesses** method retrieves list of processes running on the server machine. Only the sessions that the user has permission to query will be retrieved.

```
BOOLEAN RpcWinStationGetAllProcesses(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] ULONG Level,
  [in, out] BOUNDED_ULONG* pNumberOfProcesses,
  [out, size_is(,*pNumberOfProcesses)]
    PTS_ALL_PROCESSES_INFO* ppTsAllProcessesInfo
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationGetAllProcesses** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Level: MUST be 0. Any other value will lead to STATUS_NOT_IMPLEMENTED being returned in pResult, and FALSE returned by the method.

pNumberOfProcesses: The number of processes requested by the caller. On return, this indicates the number of processes actually stored in the *ppTsAllProcessesInfo* parameter.

ppTsAllProcessesInfo: Pointer to an array of processes allocated and returned by the method. ***ppTsAllProcessesInfo** is allocated by the method to be an array of **TS_ALL_PROCESSES_INFO** structures. The pointer returned in **ppTsAllProcessesInfo** SHOULD be freed by the caller using WinStationFreeGAPMemory.

Return Values: Returns TRUE if the call succeeded, and FALSE if the lookup failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.34 RpcWinStationGetProcessSid (Opnum 44)

The **RpcWinStationGetProcessSid** method retrieves the process security identifier (SID) for a given process ID and process start time combination.

```
BOOLEAN RpcWinStationGetProcessSid(
  [in] SERVER_HANDLE hServer,
  [in] DWORD dwUniqueProcessId,
  [in] LARGE_INTEGER ProcessStartTime,
  [out] LONG* pResult,
  [in, out, unique, size_is(dwSidSize)]
    PBYTE pProcessUserSid,
  [in, range(0, 1024 )] DWORD dwSidSize,
  [in, out] DWORD* pdwSizeNeeded
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

dwUniqueProcessId: The process ID to retrieve the SID.

ProcessStartTime: The start time of the process indicated by *dwUniqueProcessId*. *ProcessStartTime* combined with *dwUniqueProcessId* is used to identify a process.

pResult: Failure error code if the call to **RpcWinStationGetProcessSid** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

pProcessUserSid: The buffer into which the method SHOULD copy the **SID** of the process. MUST be NULL if **dwSidSize** is zero, in which case the correct size will be returned in **pdwSizeNeeded**.

dwSidSize: The size of the buffer pointed to by **pProcessUserSid**. If the buffer is too small, STATUS_BUFFER_TOO_SMALL is returned in **pResult**.

pdwSizeNeeded: Indicates the length of the **SID**. If STATUS_BUFFER_TOO_SMALL is returned in **pResult**, this indicates to the caller the correct size to allocate to a buffer prior to calling the method again.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, **pResult** indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.35 RpcWinStationGetTermSrvCountersValue (Opnum 45)

The **RpcWinStationGetTermSrvCountersValue** method retrieves the current value of requested Terminal Server performance counters.

```
BOOLEAN RpcWinStationGetTermSrvCountersValue(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, range(0, 0x1000 )] DWORD dwEntries,
  [in, out, size_is(dwEntries)] PTS_COUNTER pCounter
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationGetTermSrvCountersValue** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

dwEntries: The number of performance counters to query. Indicates the size of the array pointed to by pCounter.

pCounter: An array of **TS_COUNTER** structures. The caller MUST set the dwCounterId in the **TS_COUNTER** structures for each entry in the array to indicate the counter of which to retrieve the current value. On return, the method MUST set the value for that performance counter, or it will set the dwCounterId to 0 and bResult to 0, if the performance counter ID is not recognized or is not supported.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code. Individual entries in the array pCounter will indicate if the counter data for that counter could be retrieved or not.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.36 RpcWinStationReInitializeSecurity (Opnum 46)

The **RpcWinStationReInitializeSecurity** method reinitializes security for all non-console WinStation (remote connection protocols) specified in the registry under HKLM\System\CurrentControlSet\Terminal Server\WinStations. Existing sessions will not be affected, but future sessions will have the new security descriptor read from the registry applied to them. This method MUST be called by processes running as SYSTEM or as an Administrator.<82>

```
BOOLEAN RpcWinStationReInitializeSecurity(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationReInitializeSecurity** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.37 RpcWinStationBroadcastSystemMessage (Opnum 47)

The **RpcWinStationBroadcastSystemMessage** method sends a message to a specified session running on the Terminal Server. It performs the equivalent of BroadcastSystemMessage, sending a message to a session. The caller MUST be SYSTEM or Administrator, and this method cannot be called remotely. The session MUST be in Active or Disconnected state.<83>

```
LONG RpcWinStationBroadcastSystemMessage(
  [in] SERVER_HANDLE hServer,
  [in] ULONG sessionID,
  [in] ULONG timeOut,
  [in] DWORD dwFlags,
  [in, out, ptr] DWORD* lpdwRecipients,
  [in] ULONG uiMessage,
  [in] UINT_PTR wParam,
  [in] LONG_PTR lParam,
  [in, size_is(bufferSize)] PBYTE pBuffer,
  [in, range(0, 0x8000 )] ULONG bufferSize,
  [in] BOOLEAN fBufferHasValidData,
  [out] LONG* pResponse
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

sessionID: The session to broadcast to.

timeOut: The amount of time, in milliseconds, to wait to get a response.

dwFlags: Option flags. These are the same values as the standard method BroadcastSystemMessage (for more information, see [WININTERNALS]). BSF_POSTMESSAGE (0x00000010) is not supported. MUST be a bitwise OR of the following values.

Value	Meaning
BSF_ALLOWSFW 0x00000080	Enables the recipient to set the foreground window while processing the message.
BSF_FLUSHDISK 0x00000004	Flush the disk after each recipient processes the message.
BSF_FORCEIFHUNG 0x00000020	Continue to broadcast the message, even if the time-out period elapses or one of the recipients is hung.
BSF_IGNORECURRENTTASK 0x00000002	Do not send the message to windows that belong to the current task. This prevents an application from receiving its own message.
BSF_NOHANG 0x00000008	Force a hung application to time out. If one of the recipients times out, do not continue broadcasting the message.
BSF_NOTIMEOUTIFNOTHUNG 0x00000040	Wait for a response to the message as long as the recipient is not hung. Do not time out.
BSF_QUERY 0x00000001	Send the message to one recipient at a time, sending to a subsequent recipient only if the current recipient returns TRUE.

lpdwRecipients: Pointer to a variable that contains and receives information on the recipients of the message. These are the same values as the standard method BroadcastSystemMessage (for more information, see [WININTERNALS]). The variable can be a combination of the following values.

Value	Meaning
BSM_ALLCOMPONENTS 0x00000000	Broadcast to all system components.
BSM_ALLDESKTOPS 0x00000010	Broadcast to all desktops. Requires the SE_TCB_NAME privilege.<84>
BSM_APPLICATIONS 0x00000008	Broadcast to applications.

When the method returns, this variable receives a combination of these values, identifying what recipients actually received the message. If this parameter is NULL, the method broadcasts to all components.

uiMessage: The message to send.

wParam: The first message parameter, as defined for BroadcastSystemMessage. For more information, see [MSDN-BCSYSMSG].

lParam: The second message parameter, as defined for BroadcastSystemMessage. For more information, see [MSDN-BCSYSMSG].

pBuffer: The buffer to copy and use instead of *lParam* if the *lParam* parameter for the message type is actually a pointer.

bufferSize: The size of *pBuffer*, in bytes.

fBufferHasValidData: Set to TRUE if *pBuffer* should be used instead of *lParam*, or FALSE otherwise.

pResponse: Pointer to a variable to receive the message response. If dwFlags is BSF_QUERY, and at least one recipient returned BROADCAST_QUERY_DENY to the message, NULL is returned in pResponse.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.38 RpcWinStationSendWindowMessage (Opnum 48)

The **RpcWinStationSendWindowMessage** method performs the equivalent of SendMessage to a window on a specific session. This MUST be called locally by threads running with SYSTEM or Administrator access.<85>

```
LONG RpcWinStationSendWindowMessage(
  [in] SERVER_HANDLE hServer,
  [in] ULONG sessionID,
  [in] ULONG timeOut,
  [in] ULONG hWnd,
  [in] ULONG Msg,
  [in] UINT_PTR wParam,
  [in] LONG_PTR lParam,
  [in, size_is(bufferSize)] PBYTE pBuffer,
  [in, range(0, 0x8000 )] ULONG bufferSize,
  [in] BOOLEAN fBufferHasValidData,
  [out] LONG* pResponse
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

sessionID: The session to send the window message to.

timeOut: The amount of time, in milliseconds, to wait to get a response.

hWnd: The handle of the window in the session to receive the message.

Msg: The message to send.

wParam: The first message parameter, as defined for SendMessage. For more information, see [MSDN-SNDMSG].

lParam: The second message parameter, as defined for SendMessage. For more information, see [MSDN-SNDMSG].

pBuffer: Pointer to buffer to copy and use instead of *lParam* if the *lParam* parameter for the message type is actually a pointer.

bufferSize: The size of the buffer pointed to by *pBuffer*, in bytes.

fBufferHasValidData: Set to TRUE if *pBuffer* should be used instead of *lParam*, or FALSE otherwise.

pResponse: Pointer to a variable to receive the message response. This value is dependent on the window message sent. For more information, see [MSDN-SNDMSG].

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.39 RpcWinStationNotifyNewSession (Opnum 49)

Supported in Windows 2000 only.

The RpcWinStationNotifyNewSession method is called by winlogon to notify the Terminal Server service of a new session. This method MUST be called by threads running as SYSTEM.<86>

```
BOOLEAN RpcWinStationNotifyNewSession(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationNotifyNewSession** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not SYSTEM.

ClientLogonId: The ID of the new session.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.40 RpcWinStationGetLanAdapterName (Opnum 53)

The **RpcWinStationGetLanAdapterName** method is provided a protocol name and a LAN adapter and, if that adapter is configured with that protocol, the LAN adapter GUID is returned as a string. The GUIDs for the network adapters implementing Terminal Server are found as subkeys under the registry key HKLM\System\CurrentControlSet\Terminal Server\lanatable, where each key is the adapter's GUID, and the DWORD value LanaId under that key is the lana that is matched to the parameter LanAdapter.<87>

```
BOOLEAN RpcWinStationGetLanAdapterName(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, range(0, 0x1000 )] DWORD PdNameSize,
  [in, size_is(PdNameSize)] PWCHAR pPdName,
  [in, range(0, 1024)] ULONG LanAdapter,
  [out] ULONG* pLength,
  [out, size_is(,*pLength)] PWCHAR* ppLanAdapter
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationGetLanAdapterName** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

PdNameSize: The size, in bytes, of pPdName.

pPdName: The protocol driver type on which to retrieve information. This MUST be any of the following strings: tcp, netbios, ipx, spx. Any other value will lead to STATUS_INVALID_PARAMETER being returned in pResult.

LanAdapter: The number of the LAN adapter to retrieve information (also known as lana). 0 indicates to retrieve the LAN adapter name to indicate all LAN adapters configured with the protocol pPdName.

pLength: Pointer to the length of data in ppLanAdapter. If LanAdapter is 0, this value MUST be DEVICENAME_LENGTH+1.

ppLanAdapter: Pointer to a string allocated by this method for retrieving the LAN adapter's GUID as a string. This memory is freed by the caller using WinStationFreeMemory.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01	Successful completion.

Return value/code	Description
TRUE	
0x00 FALSE	Method call failed.

3.6.4.1.41 RpcWinStationUpdateUserConfig (Opnum 54)

The **RpcWinStationUpdateUserConfig** method requests the server to update the user configuration data for the user logged on in a specific session. This method opens the user's profile, retrieves the policy data, and overrides any data found in the user session's USERCONFIGW structure. This API MUST be called locally by threads in winlogon processes running as SYSTEM.<88>

```
BOOLEAN RpcWinStationUpdateUserConfig(
  [in] SERVER_HANDLE hServer,
  [in] DWORD ClientLogonId,
  [in] DWORD ClientProcessId,
  [in] DWORD UserToken,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

ClientLogonId: The ID of the session to update the user config for.

ClientProcessId: The process ID that is making this request. MUST be the winlogon process in the session specified by the *ClientLogonId* parameter.

UserToken: The user's token.

pResult: The failure error code if the call to **RpcWinStationUpdateUserConfig** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not SYSTEM.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.42 RpcWinStationQueryLogonCredentials (Opnum 55)

The **RpcWinStationQueryLogonCredentials** method queries the autologon credentials for a session. This API MUST be called locally by running with SYSTEM credentials.<89>

```
BOOLEAN RpcWinStationQueryLogonCredentials(
  [in] SERVER_HANDLE hServer,
  [in] ULONG LogonId,
  [out, size_is(,*pcbCredentials)]
    PCHAR* ppCredentials,
  [in, out] ULONG* pcbCredentials
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

LogonId: The ID of the session to retrieve the logon credentials for.

ppCredentials: Pointer to a structure allocated by this method for retrieving the logon credentials for the session identified by LogonId. This structure is of type WLX_CLIENT_CREDENTIALS_INFO_V2_0; for more information, see [MSDN-WLX_CCINFOV20]. This memory is freed by the caller using LocalFree.

pcbCredentials: Pointer to a variable that will receive the size of the buffer returned in ppCredentials.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.43 RpcWinStationRegisterConsoleNotification (Opnum 56)

The **RpcWinStationRegisterConsoleNotification** method registers a window in a console session to receive session notifications via the windows message WM_WTSSESSION_CHANGE. This method MUST be called from the local machine. The caller MUST be in the same session as SessionId. **RpcWinStationUnregisterConsoleNotification** SHOULD be called when the window no longer needs to receive the notifications.<90>

```
BOOLEAN RpcWinStationRegisterConsoleNotification(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] ULONG SessionId,
  [in] ULONG_PTR hWnd,
  [in] DWORD dwFlags,
  [in] DWORD dwMask
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: The failure error code if the call to **RpcWinStationRegisterConsoleNotification** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_ACCESS_DENIED 0xC0000022	Called remotely.
STATUS_INVALID_PARAMETER 0xC000000D	Caller is not in the same session as SessionId.

SessionId: The session to receive the notifications.

hWnd: The handle of the window to receive the notifications.

dwFlags: Flags.

Value	Meaning
NOTIFY_FOR_ALL_SESSIONS 1	Receive notifications for all sessions.
NOTIFY_FOR_THIS_SESSION 0	Receive notifications for just this session.

dwMask: The bitmask of the wanted notifications. This can be any combination of the following flags.

Value	Meaning
WTS_CONSOLE_CONNECT_MASK 0x00000001	A console session has connected.
WTS_CONSOLE_DISCONNECT_MASK 0x00000002	A console session has disconnected.
WTS_REMOTE_DISCONNECT_MASK 0x00000008	A remote session disconnected.
WTS_REMOTE_CONNECT_MASK 0x00000004	A remote session connected.
WTS_SESSION_LOGON_MASK 0x00000010	A user has logged on to a session.
WTS_SESSION_LOCK_MASK 0x00000040	A session is locked.
WTS_SESSION_UNLOCK_MASK 0x00000080	A session has unlocked.
WTS_SESSION_REMOTE_CONTROL_MASK 0x00000100	A session is being shadowed.

Value	Meaning
WTS_ALL_NOTIFICATION_MASK 0xFFFFFFFF	All notifications.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.44 RpcWinStationUnRegisterConsoleNotification (Opnum 57)

The **RpcWinStationUnRegisterConsoleNotification** method unregisters a window in a specific session from receiving session notifications via the windows message WM_WTSSESSION_CHANGE. The window in this session must have been previously registered to receive session notifications. This method MUST be called from the local machine. The caller MUST be in the same session as SessionId.<91>

```
BOOLEAN RpcWinStationUnRegisterConsoleNotification(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] ULONG SessionId,
  [in] ULONG hWnd
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: The failure error code if the call to **RpcWinStationUnRegisterConsoleNotification** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	The call failed remotely.
STATUS_INVALID_PARAMETER 0xC000000D	The caller is not in the session identified by *SessionId*.

SessionId: The session to stop receiving the notifications.

hWnd: The handle of the window to stop receiving the notifications.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.45 RpcWinStationUpdateSettings (Opnum 58)

The **RpcWinStationUpdateSettings** method rereads settings for all WinStation, either locally from the registry or from the session directory.<92>

```
BOOLEAN RpcWinStationUpdateSettings(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD SettingsClass,
  [in] DWORD SettingsParameters
);
```

hServer: Ignored.

pResult: Failure error code if the call to **RpcWinStationUpdateSettings** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	The caller does not have permission.
STATUS_INVALID_PARAMETER 0xC000000D	Unrecognized *SettingsClass*.

SettingsClass: The class to update settings for.

Value	Meaning
WINSTACFG_SESSDIR 0x00000001	Contacts Session Directory to reread the WinStation settings. If running in Remote Desktop mode (Windows XP) or in Remote Admin mode, this does nothing.
WINSTACFG_LEGACY 0x00000000	Rereads settings from the local registry for the configured winstations. This does the same thing as **RpcWinStationReadRegistry**.

SettingsParameters: MUST be 0.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.46 RpcWinStationShadowStop (Opnum 59)

The **RpcWinStationShadowStop** method stops all shadow operations on the specified session, including whether the session is acting as a shadow client or as a shadow target. Caller MUST have WINSTATION_DISCONNECT and WINSTATION_RESET permissions.<93>

```
BOOLEAN RpcWinStationShadowStop(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] BOOLEAN bWait
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationShadowStop** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_CTX_WINSTATION_NOT_FOUND 0xC00A0015	Indicates the session does not exist.
STATUS_CTX_SHADOW_NOT_RUNNING 0xC00A0036	Indicates the session is either not active or not being shadowed.
STATUS_ACCESS_DENIED 0xC0000022	Indicates the caller does not have permission to end shadowing on the session.

LogonId: The ID of the session on which to stop shadowing operations.

bWait: Specifies whether to wait for reset to complete. Not used.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.47 RpcWinStationCloseServerEx (Opnum 60)

The **RpcWinStationCloseServerEx** method closes the server handle for WinStation APIs.

```
BOOLEAN RpcWinStationCloseServerEx(
  [in, out] SERVER_HANDLE_EXCLUSIVE* phServer,
  [out] DWORD* pResult
);
```

phServer: Pointer to a variable that is a handle to the server. The variable is of type **SERVER_HANDLE_EXCLUSIVE**. The handle MUST be returned from a previous call to **RpcWinStationOpenServer**. On return from this method, *phServer is set to NULL.

pResult: Failure error code if the call to **RpcWinStationCloseServerEx** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Return Values: Returns TRUE if the call succeeded, and FALSE if the call failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.48 RpcWinStationIsHelpAssistantSession (Opnum 61)

The **RpcWinStationIsHelpAssistantSession** method determines if a session is created by the built-in HelpAssistant user account (used for Remote Assistance functionality). Only supported for Terminal Server protocol RDP and for sessions that aren't the console session.<94>

```
BOOLEAN RpcWinStationIsHelpAssistantSession(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] ULONG SessionId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationIsHelpAssistantSession** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_UNSUCCESSFUL 0xC0000001	The session designated by *SessionId* does not exist.

Value	Meaning
STATUS_CTX_WINSTATION_NOT_FOUND 0xC00A0015	The *SessionId* does not exist.
STATUS_WRONG_PASSWORD 0xC000006A	This is not a Help Assistant session.

SessionId: The ID of the session to check.

Return Values: Returns TRUE if the session is running as HelpAssistant, and FALSE if this is not a HelpAssistant session or if an error was encountered during the test. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.49 RpcWinStationGetMachinePolicy (Opnum 62)

the **RpcWinStationGetMachinePolicy** method returns a copy of the Terminal Server machine policy to the caller.<95>

```
BOOLEAN RpcWinStationGetMachinePolicy(
  [in] SERVER_HANDLE hServer,
  [in, out, size_is(bufferSize)]
    PBYTE pPolicy,
  [in, range(0, 0x8000 )] ULONG bufferSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pPolicy: Pointer to a buffer to receive the machine policy. This buffer should be of type POLICY_TS_MACHINE.

bufferSize: Size of the buffer pointed to by pPolicy. This size MUST be less than sizeof(POLICY_TS_MACHINE).

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.50 RpcWinStationCheckLoopBack (Opnum 65)

The **RpcWinStationCheckLoopBack** method checks if there is a loopback when a client tries to connect. Loopback refers to opening a Terminal Server session on the local machine.<96>

```
BOOLEAN RpcWinStationCheckLoopBack(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId,
  [in] DWORD TargetLogonId,
  [in, size_is(NameSize)] PWCHAR pTargetServerName,
  [in, range(0, 1024 )] DWORD NameSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationCheckLoopBack** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

STATUS_ACCESS_DENIED if a loop is detected.

ClientLogonId: The ID of the session from which the Terminal Server client was started.

TargetLogonId: The session ID to which the client is trying to connect.

pTargetServerName: The name of the target server to which the client is connecting. Must be NULL-terminated.

NameSize: The size of pTargetServerName, in characters. Includes the NULL terminator.

Return Values: Returns TRUE if there is no loopback, and FALSE if there is a loopback or if an error was encountered during the loopback test. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.51 RpcConnectCallback (Opnum 66)

The RpcConnectCallback method initiates a connection back to the Remote Assistance (RA) client. For more information, see [MSDN-RA]. This is only used on Windows XP and Windows Server 2003 for Remote Assistance scenarios. This method MUST be called by processes running as SYSTEM. Note that this function assumes that the address being passed in is an IPv4 address. IPv6 addresses are not supported.<97>

```
BOOLEAN RpcConnectCallback(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD TimeOut,
```

```
  [in] ULONG AddressType,
  [in, size_is(AddressSize)] PBYTE pAddress,
  [in, range(0, 0x1000 )] ULONG AddressSize
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationWaitSystemEvent** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_NOT_SUPPORTED 0xC00000BB	*AddressType* is not TDI_ADDRESS_TYPE_IP. For more information, see [MSDN-TDIADDRESS].
STATUS_INVALID_PARAMETER 0xC000000D	*AddressSize* is not TDI_ADDRESS_LENGTH_IP. For more information, see [MSDN-TDIADDRESS].
STATUS_ACCESS_DENIED 0xC0000022	The caller is not SYSTEM.

TimeOut: Time out, in seconds, to wait for this connection to succeed.

AddressType: MUST be TDI_ADDRESS_TYPE_IP. For more information, see [MSDN-TDIADDRESS].

pAddress: Pointer to the address itself. MUST be TDI_ADDRESS_IP. For more information, see [MSDN-TDIADDRESS].

AddressSize: MUST be TDI_ADDRESS_LENGTH_IP. For more information, see [MSDN-TDIADDRESS].

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.52 RpcRemoteAssistancePrepareSystemRestore (Opnum 69)

The **RpcRemoteAssistancePrepareSystemRestore** method prepares Terminal Server service for a Remote Assistance-specific system restore. This includes Remote Assistance-specific encryption key or registry settings that need to be preserved. This method MUST be called by a local Administrator.<98>

```
BOOLEAN RpcRemoteAssistancePrepareSystemRestore(
```

```
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcRemoteAssistancePrepareSystemRestore** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	Called remotely or called if the caller is not a local Administrator.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.53 RpcWinStationGetAllProcesses_NT6 (Opnum 70)

The **RpcWinStationGetAllProcesses_NT6** method retrieves the processes running a remote server on which the caller has access to retrieve information.

```
BOOLEAN RpcWinStationGetAllProcesses_NT6(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] ULONG Level,
  [in, out] BOUNDED_ULONG* pNumberOfProcesses,
  [out, size_is(,*pNumberOfProcesses)]
    TS_ALL_PROCESSES_INFO_NT6* ppTsAllProcessesInfo
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationGetAllProcesses_NT6** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Level: Must be GAP_LEVEL_BASIC (0). Any other value will lead to STATUS_NOT_IMPLEMENTED returned in pResult, and FALSE returned by the method.

pNumberOfProcesses: The number of processes requested by the caller. On return, this indicates the number of processes actually stored in the ppTsAllProcessesInfo parameter.

ppTsAllProcessesInfo: Pointer to an array of processes allocated and returned by the method. *ppTsAllProcessesInfo is allocated by the method to be an array of **TS_ALL_PROCESSES_INFO_NT6** structures. The pointer returned in *ppTsAllProcessesInfo* should be freed by the caller using WinStationFreeGAPMemory.

Return Values: Returns TRUE if the call succeeded, and FALSE if the lookup failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.54 RpcWinStationRegisterNotificationEvent (Opnum 71)

The **RpcWinStationRegisterNotificationEvent** method is similar to **RpcWinStationRegisterConsoleNotification** except that instead of registering a window (to receive a window handle instead), an event handle is passed in to be signaled when a notification should be sent. This method cannot be called remotely. The caller must be in the same session as SessionId. **RpcWinStationUnregisterNotificationEvent** SHOULD be called when the process no longer needs to receive the notifications.<99>

```
BOOLEAN RpcWinStationRegisterNotificationEvent(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [out] REGISTRATION_HANDLE* pNotificationId,
  [in] ULONG_PTR EventHandle,
  [in] DWORD TargetSessionId,
  [in] DWORD dwMask,
  [in] DWORD dwProcessId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationRegisterNotificationEvent** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	Called remotely.
STATUS_INVALID_PARAMETER 0xC000000D	The caller is not in the same session as *SessionId*.

pNotificationId: The notification ID to be passed by the caller to **RpcWinStationUnregisterNotificationEvent** when unregistering notifications.

EventHandle: The event handle to signal when the notifications need to be sent.

TargetSessionId: The session to receive the notifications.

dwMask: The bitmask of the wanted notifications. This can be any combination of the following flags.

Value	Meaning
WTS_CONSOLE_CONNECT_MASK 0x00000001	A console session has connected.
WTS_CONSOLE_DISCONNECT_MASK 0x00000002	A console session has disconnected.
WTS_REMOTE_DISCONNECT_MASK 0x00000008	A remote session disconnected.
WTS_REMOTE_CONNECT_MASK 0x00000004	A remote session connected.
WTS_SESSION_LOGON_MASK 0x00000010	A user has logged on to a session.
WTS_SESSION_LOCK_MASK 0x00000040	A session is locked.
WTS_SESSION_UNLOCK_MASK 0x00000080	A session has unlocked.
WTS_SESSION_REMOTE_CONTROL_MASK 0x00000100	A session is being shadowed.
WTS_ALL_NOTIFICATION_MASK 0xFFFFFFFF	All notifications.

dwProcessId: The process to receive the notifications. Caller must have access to the process.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.55 RpcWinStationUnRegisterNotificationEvent (Opnum 72)

The **RpcWinStationUnRegisterNotificationEvent** method unregisters an event previously registered by **RpcWinStationUnregisterNotificationEvent** to be signaled when notifications take place. This function is similar to **RpcWinStationUnRegisterConsoleNotification**. This method cannot be called remotely. The caller must be in the same session as SessionId.<100>

```
BOOLEAN RpcWinStationUnRegisterNotificationEvent(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, out] REGISTRATION_HANDLE* NotificationId
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationUnregisterNotificationEvent** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	the call was successful.
STATUS_ACCESS_DENIED 0xC0000022	Called remotely.
STATUS_INVALID_PARAMETER 0xC000000D	The caller is not in the session identified by *SessionId*.

NotificationId: The notification ID returned previously by **RpcWinStationUnregisterNotificationEvent**.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.56 RpcWinStationAutoReconnect (Opnum 73)

The **RpcWinStationAutoReconnect** method auto-reconnects a disconnected Terminal Server client to a given session.<101>

This method MUST be called internally by local processes running as SYSTEM.

```
BOOLEAN RpcWinStationAutoReconnect(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD LogonId,
  [in] DWORD flags
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationAutoReconnect** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	The non-system processes attempted a call, or the session does not exist.
STATUS_INVALID_PARAMETER 0xC000000D	Attempt to call this method remotely.
STATUS_NOT_FOUND 0xC0000225	No auto-reconnect information specified.
STATUS_CTX_WINSTATION_BUSY 0xC00A0017	The *flags* are nonzero.

LogonId: The identifier of the session to reconnect to.

flags: Unused, and MUST be zero.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.57 RpcWinStationCheckAccess (Opnum 74)

The **RpcWinStationCheckAccess** method determines if a user has specific permissions on a session. This method MUST be called by a process running as SYSTEM.<102>

```
BOOLEAN RpcWinStationCheckAccess(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in] DWORD ClientLogonId,
  [in] DWORD UserToken,
  [in] ULONG LogonId,
  [in] ULONG AccessMask
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationCheckAccess** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not running as SYSTEM.
STATUS_CTX_WINSTATION_NOT_FOUND 0xC00A0015	One of the sessions does not exist.

ClientLogonId: The session ID of the user on which to check permissions.

UserToken: The user's token.

LogonId: The ID of the session to check user's permissions against.

AccessMask: The bitmask of the desired access. This MUST be any bitwise OR of the following values (for more information, see [WININTERNALS]).

Name	Value
WINSTATION_QUERY	0x00000001
WINSTATION_SET	0x00000002
WINSTATION_RESET	0x00000004
WINSTATION_VIRTUAL	0x00000008
WINSTATION_SHADOW	0x00000010
WINSTATION_LOGON	0x00000020
WINSTATION_LOGOFF	0x00000040
WINSTATION_MSG	0x00000080
WINSTATION_CONNECT	0x00000100
WINSTATION_DISCONNECT	0x00000200
WINSTATION_GUEST_ACCESS	0x00000020
WINSTATION_CURRENT_GUESS_ACCESS	0x00000048
WINSTATION_USER_ACCESS	0x00000121
WINSTATION_CURRENT_USER_ACCESS	0x0000024E
WINSTATION_ALL_ACCESS	0x000F03BF
STANDARD_RIGHTS_REQUIRED	0x000F0000

Return Values: Returns TRUE if the user has the permissions, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.6.4.1.58 RpcWinStationOpenSessionDirectory (Opnum 75)

The **RpcWinStationOpenSessionDirectory** method pings the Session Directory to see if it can accept RPC calls. The caller must be either SYSTEM or an Administrator.<103>

```
BOOLEAN RpcWinStationOpenSessionDirectory(
  [in] SERVER_HANDLE hServer,
  [out] DWORD* pResult,
  [in, string, max_is(64)] PWCHAR pszServerName
);
```

hServer: Handle to the server object. This is of type **SERVER_HANDLE**. The *hServer* argument MUST be obtained from a previous call to **RpcWinStationOpenServer**.

pResult: Failure error code if the call to **RpcWinStationOpenSessionDirectory** failed. If the call was successful, this parameter MUST be STATUS_SUCCESS (0x00000000).

Value	Meaning
STATUS_SUCCESS 0x00000000	The call was successful.
STATUS_UNSUCCESSFUL 0xC0000001	The server is not in application server mode on an advanced servers SKU.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not SYSTEM nor an Administrator.

pszServerName: The name of the server hosting session directory to attempt to connect to.

Return Values: Returns TRUE if the call succeeded, and FALSE if the method failed. On failure, pResult indicates the failure status code.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.7 Terminal Server Licensing Client Details

3.7.1 Abstract Data Model

No abstract data model is required.

3.7.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE].

3.7.3 Initialization

The client MUST create an RPC connection to the Terminal Server, using the details specified in section 2.1.

3.7.4 Message Processing Events and Sequencing Rules

This is a stateless protocol. No sequence of method calls is imposed.

When a method completes, the values returned by RPC MUST be returned unmodified to the upper layer.

The client MUST ignore errors returned from the RPC server and notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.

3.7.5 Timer Events

There are no timer events.

3.7.6 Other Local Events

There are no local events.

3.8 Terminal Server Licensing Server Details

3.8.1 Abstract Data Model

No abstract data model is required.

3.8.2 Timers

No timers are required by this protocol.

3.8.3 Initialization

Parameters necessary to initialize the RPC protocol are specified in section 2.1.

3.8.4 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. For more information, see [MS-RPCE] section 1.3.

All methods implemented by the Terminal Server server SHOULD enforce appropriate security measures to make sure the Terminal Server client has required permissions to execute the method. All methods MUST be RPC asynchronous calls.

The methods MAY throw an exception, and the Terminal Server client MUST handle these exceptions gracefully.

3.8.4.1 LCRPC

The **LCRPC** interface provides methods that open, close, and manipulate information on licensing. The version for this interface is 1.0.

For endpoints, UUID values, and versions, see sections 2.1 and 1.9.

Methods in RPC Opnum Order

Method	Description
RpcLicensingOpenServer	Returns a handle to the Terminal Server that can be used to query or modify licensing policies of the server. Opnum: 0
RpcLicensingCloseServer	Closes the handle to the Terminal Server returned by **RpcLicensingOpenServer**. Opnum: 1
Opnum2NotUsedOnWire	Not implemented. Opnum: 2
Opnum3NotUsedOnWire	Not implemented. Opnum: 3
RpcLicensingSetPolicy	Sets the licensing policies for the Terminal Server. Opnum: 4
RpcLicensingGetAvailablePolicyIds	Returns a list of licensing policy IDs that can be applied to the Terminal Server. Opnum: 5
RpcLicensingGetPolicy	Returns the current licensing policy used on the Terminal Server. Opnum: 6
RpcLicensingGetPolicyInformation	Returns detailed information on the currently used licensing policy on the Terminal Server. Opnum: 7
Opnum8NotUsedOnWire	Not implemented. Opnum: 8
RpcLicensingServerPing	Sends a ping message to the licensing policy manager for the session belonging to the user that is making the call and running on the Terminal Server. Opnum: 9
RpcGetSessionUnderArbitration	Returns the current session, if any, that is under arbitration on the Terminal Server. Opnum: 10

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<104> because it does not affect interoperability.

3.8.4.1.1 RpcLicensingOpenServer (Opnum 0)

The **RpcLicensingOpenServer** method returns a handle to the Terminal Server, which can then be used to query or modify licensing policies of the server.

```
BOOLEAN RpcLicensingOpenServer(
  [in] handle_t hBinding,
  [out] LCRPC_HANDLE* phServer,
  [out, ref] PLONG pStatus
);
```

hBinding: The RPC binding handle. For more information, see [MSDN-RPCBIND].

phServer: Handle to the Terminal Server.

pStatus: The status of the call. Returns STATUS_SUCCESS if call succeeded; otherwise, it returns an error value.

Return Values: The method MUST return TRUE on success; otherwise, it returns FALSE.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.8.4.1.2 RpcLicensingCloseServer (Opnum 1)

The **RpcLicensingCloseServer** method closes the handle to the Terminal Server returned by **RpcLicensingOpenServer**. This method MUST be called after **RpcLicensingOpenServer**.

```
void RpcLicensingCloseServer(
  [in, out] LCRPC_HANDLE_EXCLUSIVE* phServer
);
```

phServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

Return Values: The method does not return a value.

3.8.4.1.3 RpcLicensingSetPolicy (Opnum 4)

The **RpcLicensingSetPolicy** method sets the licensing policies for the Terminal Server. This method MUST be called by processes running as an Administrator.

```
LONG RpcLicensingSetPolicy(
  [in] LCRPC_HANDLE hServer,
  [in] ULONG ulPolicyId,
  [out, ref] PLONG pNewPolicyStatus
);
```

hServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

ulPolicyId: The ID of the policy to apply. The following IDs are supported.

Value	Meaning
DEFAULT_POLICY_ID 1	This is supported only when the Terminal Server is in Remote Administration mode.
PERSEAT_POLICY_ID 2	This is supported only when the Terminal Server is in Application Server mode.
INTCONN_POLICY_ID 3	This is supported only when the Terminal Server is in Application Server mode and supported for Windows 2000 Server only.
PERUSER_POLICY_ID 4	This is supported only when the Terminal Server is in Application Server mode and supported on Windows Server 2003 and later.
POLICY_NOT_CONFIGURED 5	This is not supported for this call.

pNewPolicyStatus: The status of setting the new policy. Returns STATUS_SUCCESS if the call succeeded; otherwise, it returns an error value.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

Return value/code	Description
0x00000000 STATUS_SUCCESS	Successful completion.

3.8.4.1.4 RpcLicensingGetAvailablePolicyIds (Opnum 5)

The **RpcLicensingGetAvailablePolicyIds** method returns a list of licensing policy IDs that can be applied to the Terminal Server. This method MUST be called after **RpcLicensingOpenServer**.

```
BOOLEAN RpcLicensingGetAvailablePolicyIds(
  [in] LCRPC_HANDLE hServer,
  [out, size_is(,*pcPolicies)] PULONG* ppulPolicyIds,
  [out, ref] PULONG pcPolicies,
  [out, ref] PLONG pStatus
);
```

hServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

ppulPolicyIds: An array of policy IDs that can be applied to the Terminal Server.

pcPolicies: The number of policy IDs returned.

pStatus: The status of the call. Returns STATUS_SUCCESS if call succeeded; otherwise, it returns an error value.

Return Values: The method MUST return TRUE on success; otherwise, it returns FALSE.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.8.4.1.5 RpcLicensingGetPolicy (Opnum 6)

The **RpcLicensingGetPolicy** method returns the current licensing policy used on the Terminal Server. This method MUST be called after **RpcLicensingOpenServer**.

```
BOOLEAN RpcLicensingGetPolicy(
  [in] LCRPC_HANDLE hServer,
  [out, ref] PULONG pulPolicyId,
  [out, ref] PLONG pStatus
);
```

hServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

pulPolicyId: The ID of the currently used licensing policy on the server.

pStatus: The status of the call. Returns STATUS_SUCCESS (0x00000000) if the call succeeded; otherwise, it returns an error value.

Return Values: The method MUST return TRUE on success; otherwise, it returns FALSE.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.8.4.1.6 RpcLicensingGetPolicyInformation (Opnum 7)

The **RpcLicensingGetPolicyInformation** method returns detailed information on the currently used licensing policy on the Terminal Server. This method MUST be called after **RpcLicensingOpenServer**.

```
BOOLEAN RpcLicensingGetPolicyInformation(
  [in] LCRPC_HANDLE hServer,
  [in] ULONG ulPolicyId,
  [in, out, ref] PULONG pulVersion,
  [out, size_is(,*pcbPolicyInfo)]
    PCHAR* ppPolicyInfo,
  [in, out, ref] PULONG pcbPolicyInfo,
  [out, ref] PLONG pStatus
);
```

hServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

ulPolicyId: The ID of the currently used licensing policy on the server.

pulVersion: The version of the policy being used. The only currently supported value is LCPOLICYINFOTYPE_V1.

ppPolicyInfo: Detailed information on the policy. This data is of the type LCPOLICYINFO_V1.

pcbPolicyInfo: The size of the *ppPolicyInfo*, in bytes.

pStatus: The status of the call. Returns STATUS_SUCCESS if call succeeded; otherwise, it returns an error value.

Return Values: The method MUST return TRUE on success; otherwise, it returns FALSE.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.8.4.1.7 RpcLicensingServerPing (Opnum 9)

The **RpcLicensingServerPing** method sends a ping message to the licensing policy manager for the session belonging to the user that is making the call and running on the Terminal Server. This method MUST be called after **RpcLicensingOpenServer**.

```
BOOLEAN RpcLicensingServerPing(
  [in] LCRPC_HANDLE hServer,
  [out, ref] PLONG pStatus
);
```

hServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

pStatus: The status of the ping call. Returns STATUS_SUCCESS if call succeeded; otherwise, it returns an error value. A success verifies that the licensing policy monitor for the session belonging to the caller of this function is still active.

Return Values: The method MUST return TRUE on success; otherwise, it returns FALSE.

Return value/code	Description
0x01 TRUE	Successful completion.
0x00 FALSE	Method call failed.

3.8.4.1.8 RpcGetSessionUnderArbitration (Opnum 10)

The **RpcGetSessionUnderArbitration** method returns the current session, if any, which is under arbitration on the Terminal Server. Arbitration is the process used by the Terminal Server to determine whether to create a new session for the client connecting to the Terminal Server or to connect it to an existing session. This method MUST be called after **RpcLicensingOpenServer**.

```
HRESULT RpcGetSessionUnderArbitration(
  [in] LCRPC_HANDLE hServer,
  [in] GUID* pTerminalType,
  [out, ref] PULONG pSessionId
);
```

hServer: Handle to the Terminal Server returned by **RpcLicensingOpenServer**.

pTerminalType: The GUID of the type of the terminal for this connection. For more information on terminal types, see section 2.2.2.4.

pSessionId: The ID of the temporary session created by the Terminal Server to handle the connection and that is currently under arbitration.

Return Values: The method MUST return S_OK on success.

Return value/code	Description
0x00000000 S_OK	Successful completion.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Terminal Services Terminal Server Runtime Interface Protocol.

4.1 LSM Enumeration Example

The following example shows how to enumerate sessions on a Terminal Server. This example uses **TermSrvBindSecure** from section 4.3.

1. Get the LSM Binding

```
HANDLE GetLSMBinding(LPWSTR pszServerName)
{
  HANDLE hLSMBinding = NULL;
  RPC_STATUS rpcStatus = RPC_S_OK;

  //ASSERT( NULL != pszServerName );
  rpcStatus = TermSrvBindSecure(
    gpszPublicUuid,
    gpszRemoteProtocolSequence,
    pszServerName,
    TSRPC_REMOTE_ENDPOINT,
    gpszOptions,
    &hLSMBinding
    );

  if( rpcStatus != RPC_S_OK || hLSMBinding == NULL)
  {
    wprintf(L"ERR: TermSrvBindSecure failed: %d\n",
            rpcStatus );
    SetLastError( rpcStatus );
  }

  return hLSMBinding;
}
```

2. Enumerate the sessions

```
RpcTryExcept
{
  hr = RpcOpenEnum( hLSMBind, &hEnum );
  if(hr == S_OK)
  {
    hr = RpcGetEnumResult( hEnum, &pAllSessions,
                    CURRENT_ENUM_LEVEL, &Entries );
    if(hr == S_OK)
    {
      for(ULONG i=0;i<Entries;i++)
      {
        wprintf(L"%-10d %-20s %-40s\n",
          pAllSessions[i].Data.SessionEnum_Level3.SessionId,
          WinstationStateClassNames[pAllSessions[i].Data.
                                    SessionEnum_Level3.State],
```

```
            pAllSessions[i].Data.SessionEnum Level3.Name);

            if( NULL != pAllSessions[i].Data.SessionEnum_Level3.
                        pProtocolData )
            {
              MIDL user free( pAllSessions[i].Data.
                           SessionEnum Level3.pProtocolData );
            }
          }

          rv1 = TRUE;
        }
        else
        {
          wprintf(L"ERR: RpcGetEnumResult failed %d\n",hr );
        }

        if ( pAllSessions )
        {
          MIDL_user_free(pAllSessions);
        }

      }
      else
      {
        wprintf(L"ERR: RpcOpenEnum failed %d\n",hr );
      }

    }

    RpcExcept(I RpcExceptionFilter(RpcExceptionCode()))
    {
      wprintf(L"ERR: RPC Exception %d\n",RpcExceptionCode() );
    }

    RpcEndExcept
```

3. Close the handles

```
if(hEnum)
  RpcCloseEnum(&hEnum);

if(hLSMBind)
  RpcBindingFree(&hLSMBind);
```

4.2 TermService Listener Example

The following example retrieves the listeners that run on the Terminal Server. This example uses **TermSrvBindSecure** from section 4.3

1. Get the RCM binding

```
HANDLE GetRCMBinding(LPWSTR pszServerName)
{
  RPC_STATUS rpcStatus = RPC_S_OK;
```

```
  HANDLE hRCMBinding = NULL;

  rpcStatus = TermSrvBindSecure(
    gpszPublicUuid,
    gpszRemoteProtocolSequence,
    pszServerName,
    TSRCMRPC_REMOTE_ENDPOINT,
    gpszOptions,
    &hRCMBinding
    );

  if( rpcStatus != RPC_S_OK  || hRCMBinding == NULL)
  {
    wprintf(L"ERR: TermSrvBindSecure failed: %d\n", rpcStatus );
    SetLastError( rpcStatus );
  }

  return hRCMBinding
}
```

2. Get the list of listeners

```
hRCMBind = GetRCMBinding(pszServerName);
if(hRCMBind)
{
    RpcTryExcept
    {
        hr = RpcGetAllListeners( hRCMBind, &pListeners,
                                 CURRENT_LST_ENUM_LEVEL, &NumListeners );
        if(hr == S_OK)
        {
            for(ULONG i=0;i<Entries;i++)
            {
                wprintf(L"%-10d %-20s %-40s\n",
                    pListeners[i].Data.ListenerEnum_Level1.Id,
WinstationStateClassNames[pListeners[i].Data.ListenerEnum_Level1.bListening ?
State Listen : State Down],
                    pListeners[i].Data.ListenerEnum Level1.Name);
            }
            rv2 = TRUE;
        }
        else
        {
            wprintf(L"ERR: RpcGetAllListeners failed %d\n",hr );
        }

        if ( pListeners )
        {
            MIDL user free( pListeners );
        }

    }

    RpcExcept(I RpcExceptionFilter(RpcExceptionCode()))
    {
        hr = HRESULT FROM WIN32( RpcExceptionCode() );
        wprintf(L"ERR: RpcGetAllListeners threw an exception: 0x%x\n",
                 hr );
```

```
        }

        RpcEndExcept


        RpcBindingFree(&hRCMBind);
    }
```

3. Close the handle

```
    if(hRCMBind)
              RpcBindingFree(&hRCMBind);
```

4.3 TermSrvBindSecure Example

The following example creates an RPC binding to an endpoint that uses authentication, authorization, and security quality-of-service information.

```
RPC_STATUS
TermSrvBindSecure(
    LPCWSTR pszUuid,
    LPCWSTR pszProtocolSequence,
    LPCWSTR pszNetworkAddress,
    LPCWSTR pszEndPoint,
    LPCWSTR pszOptions,
    RPC_BINDING_HANDLE *pHandle
    )
{
    RPC_STATUS Status;
    RPC_SECURITY_QOS qos;
    LPWSTR wszServerSPN = NULL;

    *pHandle = NULL;

    Status = TermSrvBind(
                    pszUuid,
                    pszProtocolSequence,
                    pszNetworkAddress,
                    pszEndPoint,
                    pszOptions,
                    pHandle);

    if( Status != RPC_S_OK )
    {
        wprintf ( L"Error %d in TermSrvBind", Status );
        goto TS_EXIT_POINT;
    }

    qos.Capabilities = RPC_C_QOS_CAPABILITIES_MUTUAL_AUTH;
    qos.IdentityTracking = RPC_C_QOS_IDENTITY_DYNAMIC;
    qos.ImpersonationType = RPC_C_IMP_LEVEL_IMPERSONATE;
```

```
    qos.Version = RPC_C_SECURITY_QOS_VERSION;

    if( PrepareServerSPN( pszNetworkAddress, &wszServerSPN ))
    {
        Status = RpcBindingSetAuthInfoEx(
                        *pHandle,
                        wszServerSPN,
                        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
                        RPC_C_AUTHN_GSS_NEGOTIATE,
                        NULL,
                        RPC_C_AUTHZ_NAME,
                        &qos);

        LocalFree(wszServerSPN);
    }
    else
    {
        Status = RpcBindingSetAuthInfoEx(
                        *pHandle,
                        (LPWSTR)pszNetworkAddress,
                        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
                        RPC_C_AUTHN_GSS_NEGOTIATE,
                        NULL,
                        RPC_C_AUTHZ_NAME,
                        &qos);
    }

    if ( RPC_S_OK != Status )
    {
        wprintf ( L"Error %d in RpcBindingSetAuthInfoEx", Status );
        goto TS_EXIT_POINT;
    }

TS_EXIT_POINT:

    if ( RPC_S_OK != Status &&
         NULL != *pHandle )
    {
        RpcBindingFree( pHandle );
    }

    return Status;
}
```

Generate a standard RPC binding from the protocol sequence, security options, and UUID, for example.

```
RPC_STATUS
TermSrvBind(
    IN LPCWSTR pszUuid,
    IN LPCWSTR pszProtocolSequence,
    IN LPCWSTR pszNetworkAddress,
    IN LPCWSTR pszEndPoint,
    IN LPCWSTR pszOptions,
```

```
    OUT RPC_BINDING_HANDLE *pHandle
    )
{
    RPC_STATUS Status;
    LPWSTR pszString = NULL;

    /*
     * Compose the binding string using the helper routine
     * and our protocol sequence, security options, UUID, etc.
     */
    Status = RpcStringBindingCompose(
                (LPWSTR)pszUuid,
                (LPWSTR)pszProtocolSequence,
                (LPWSTR)pszNetworkAddress,
                (LPWSTR)pszEndPoint,
                (LPWSTR)pszOptions,
                &pszString
                );

    if( Status != RPC_S_OK )
    {
        wprintf ( L"Error %d in RpcStringBindingCompose", Status );
        goto TS_EXIT_POINT;
    }

    /*
     * Now generate the RPC binding from the cononical RPC
     * binding string.
     */
    Status = RpcBindingFromStringBinding(
                pszString,
                pHandle
                );

    if( Status != RPC_S_OK )
    {
        wprintf ( L"Error %d in RpcBindingFromStringBinding", Status );
        goto TS_EXIT_POINT;
    }

TS_EXIT_POINT:

    if ( NULL != pszString )
    {
        /*
         * Free the memory returned from RpcStringBindingCompose()
         */
        RpcStringFree( &pszString );
    }

    return( Status );
}
```

Recreate a valid SPN for Windows Vista from an existing SPN.

```
BOOL
PrepareServerSPN(
    IN LPCWSTR pszNetworkAddress,
      deref out opt LPWSTR *ppwszServerSPN
    )
{
    // longhorn RPC does not accept "net use" credential anymore.
    // <Domain>\<Machine> is not a valid SPN, a valid SPN is host/<Machine Name>

    LPWSTR pszTemplate = L"host/%s";
    *ppwszServerSPN = NULL;
    HRESULT hr = S OK;

    UINT stringLength = wcslen(pszTemplate)+wcslen(pszNetworkAddress)+1;

    *ppwszServerSPN = (LPWSTR)LocalAlloc(LPTR, stringLength * sizeof(WCHAR));
    if(*ppwszServerSPN)
    {
        hr = StringCchPrintf( *ppwszServerSPN, stringLength, pszTemplate,
                              pszNetworkAddress );
        ASSERT( SUCCEEDED( hr ));
    }

    if( FAILED(hr) )
    {
        if( NULL != *ppwszServerSPN )
        {
            LocalFree( *ppwszServerSPN );
            *ppwszServerSPN = NULL;
        }
    }

    return SUCCEEDED(hr);
}
```

4.4 Legacy Example

The following example shows how to enumerate sessions that use the legacy RPC methods.

1. Get the binding

```
   Result = TermSrvBindSecure(
      pszUuid,
      pszRemoteProtocolSequence,
      pServerName,
      pszRemoteEndPoint,
      pszOptions,
      &RpcTSHandle
     );
     //
     // Get a context handle from the server so it can
     // manage the connections state
     //
     RpcTryExcept {
       rc = RpcWinStationOpenServer( RpcTSHandle, &Result, &ContextHandle );
     }
     RpcExcept(I RpcExceptionFilter(RpcExceptionCode())) {
       Result = RpcExceptionCode();
       rc = FALSE;
       wprintf(L"ERR: RPC Exception %d\n",Result );
```

```
  }
  RpcEndExcept
```

2. Enumerate the sessions

```
RpcTryExcept {

  rc = RpcWinStationEnumerate(
     hServer,
     &Result,
     &LogonIdCount,
     (PCHAR)pLogonId,
     &Length,
     &Index
    );
  Result = RtlNtStatusToDosError( Result );
  if ( Result == ERROR_NO_MORE_ITEMS) {
    Result = ERROR_SUCCESS;
    break;
  }

  if(rc == TRUE)
  {
    wprintf(L"SessionID  State                Name\n");
    for(ULONG i=0;i<LogonIdCount;i++)
    {
      wprintf(L"%-10d %-20s %-40s\n",
        pLogonId[i].SessionId,
            WinstationStateClassNames[pLogonId[i].State],
            pLogonId[i].WinStationName);
    }
  }
}
RpcExcept(I_RpcExceptionFilter(RpcExceptionCode())) {
 Result = RpcExceptionCode();
 wprintf(L"ERR: RPC Exception %d\n",Result );
}
RpcEndExcept
```

3. Close the binding handles

```
RpcTryExcept {

  bSuccess = RpcWinStationCloseServerEx( pHandle, pdwResult );
  if( !bSuccess ) *pdwResult = RtlNtStatusToDosError( *pdwResult );

}
RpcExcept(I_RpcExceptionFilter(RpcExceptionCode())) {
```

```
  *pdwResult = RpcExceptionCode();
  bSuccess = FALSE;
}
RpcEndExcept
```

5 Security

The following sections specify security considerations for implementers of the Terminal Services Terminal Server Runtime Interface Protocol.

5.1 Security Considerations for Implementers

The Terminal Services Terminal Server Runtime Interface Protocol allows any user to connect to the server, as specified in section 2.1. Therefore, any security bug in the server implementation could be exploitable. The server implementation should enforce security on each method.

5.2 Index of Security Parameters

Security parameter	Section
Authentication Protocol	2.1

6 Appendix A: Full IDL

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

6.1 tspubrpc.idl

For ease of implementation, the full IDL is provided, where "ms-dtyp.idl" is the IDL as specified in [MS-DTYP] Appendix A and "tsdef.h" is as specified in section 6.5.

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

#include "tsdef.h"
#include "allproc.h"

[
    uuid(484809d6-4239-471b-b5bc-61df8c23ac48),
    version(1.0),
    pointer_default(unique)
]

//
//  Public rpc interface to the session object
//

interface TermSrvSession
{
    typedef [context_handle] void *SESSION_HANDLE;
    typedef [context_handle] void *SESSION_HANDLE_EXCLUSIVE;

    typedef struct _LSMSessionInformation {
        [string] WCHAR* pszUserName;
        [string] WCHAR* pszDomain;
        [string] WCHAR* pszTerminalName;
        LONG            SessionState;
        BOOL            DesktopLocked;
        hyper           ConnectTime;
        hyper           DisconnectTime;
        hyper           LogonTime;
    } LSMSESSIONINFORMATION,
        *PLSMSESSIONINFORMATION;

    //
    // Per Session specific call
    HRESULT RpcOpenSession(
        [in]    handle_t    hBinding,
        [in]    LONG        SessionId,
        [out]   SESSION_HANDLE *phSession
        );

    HRESULT RpcCloseSession(
        [in,out] SESSION_HANDLE_EXCLUSIVE *phSession
        );

    HRESULT RpcConnect(
        [in]    SESSION_HANDLE  hSession,
```

```
    [in]    LONG            TargetSessionId,
    [in,string]    WCHAR    *szPassword
    );

HRESULT RpcDisconnect(
    [in]    SESSION_HANDLE hSession
    );

HRESULT RpcLogoff(
    [in]    SESSION_HANDLE hSession
    );

HRESULT RpcGetUserName(
    [in]    SESSION_HANDLE hSession,
    [out, string]   WCHAR   **pszUserName,
    [out, string]   WCHAR   **pszDomain
    );

HRESULT RpcGetTerminalName(
    [in]    SESSION_HANDLE hSession,
    [out, string]   WCHAR   **pszTerminalName
    );

HRESULT RpcGetState(
    [in]    SESSION_HANDLE  hSession,
    [out]   LONG            *plState
    );

HRESULT RpcIsSessionDesktopLocked(
    [in]    SESSION_HANDLE  hSession
    );

HRESULT RpcShowMessageBox(
    [in]    SESSION_HANDLE  hSession,
    [in, string] WCHAR      *szTitle,
    [in, string] WCHAR      *szMessage,
    [in]    ULONG           ulStyle,
    [in]    ULONG           ulTimeout,
    [out]   ULONG           *pulResponse,
    [in]    BOOL            bDoNotWait
    );

HRESULT RpcGetTimes(
    [in]    SESSION_HANDLE  hSession,
    [out]   hyper           *pConnectTime,
    [out]   hyper           *pDisconnectTime,
    [out]   hyper           *pLogonTime
    );

HRESULT RpcGetSessionCounters(
    [in]    handle_t            hBinding,
    [in,out, size_is(uEntries)] PTS_COUNTER pCounter,
    [in]                        ULONG       uEntries
    );

HRESULT RpcGetSessionInformation(
    [in]    handle_t    hBinding,
    [in]    LONG        SessionId,
```

```
        [ref, out] PLSMSESSIONINFORMATION pSessionInfo
        );

    HRESULT RpcSwitchToServicesSession(
        [in] handle_t hBinding
        );

    HRESULT RpcRevertFromServicesSession(
        [in] handle_t hBinding
        );

    HRESULT RpcGetLoggedOnCount(
        [in] handle_t hBinding,
        [out] ULONG *pUserSessions,
        [out] ULONG *pDeviceSessions
        );
}

//
//  notifications
//

[
    uuid(11899a43-2b68-4a76-92e3-a3d6ad8c26ce),
    version(1.0),
    pointer_default(unique)
]
interface TermSrvNotification
{
    HRESULT RpcWaitForSessionState(
        [in]        handle_t        hBinding,
        [in]        LONG            SessionId,
        [in]        LONG            State,
        [in]        ULONG           Timeout
        );

    typedef [context_handle] void *NOTIFY_HANDLE;
    typedef [context_handle] void *NOTIFY_HANDLE_EXCLUSIVE;

    HRESULT RpcRegisterAsyncNotification(
        [in]        handle_t        hBinding,
        [in]        LONG            SessionId,
        [in]        TNotificationId Mask,
        [out]       NOTIFY_HANDLE   *phNotify
        );

    HRESULT RpcWaitAsyncNotification(
        [in]        NOTIFY_HANDLE   hNotify,
        [out, size_is(, *pEntries)]
    PSESSION_CHANGE *SessionChange,
        [out]       ULONG           *pEntries
        );

    HRESULT RpcUnRegisterAsyncNotification(
        [in,out]    NOTIFY_HANDLE_EXCLUSIVE   *phNotify
        );
}
```

```
//
// enumerations
//

[
    uuid(88143fd0-c28d-4b2b-8fef-8d882f6a9390),
    version(1.0),
    pointer_default(unique)
]
interface TermSrvEnumeration
{
    typedef [context_handle] void *ENUM_HANDLE;
#define ENUM_LEVEL1          1
#define ENUM_LEVEL2          2
#define ENUM_LEVEL3          3
#define CURRENT_ENUM_LEVEL   3

    typedef struct _SESSIONENUM_LEVEL1 {
        LONG    SessionId;
        LONG    State;
        WCHAR   Name[33];
    } SESSIONENUM_LEVEL1,
        *PSESSIONENUM_LEVEL1;

    typedef struct _SESSIONENUM_LEVEL2 {
        LONG    SessionId;
        LONG    State;
        WCHAR   Name[33];
        ULONG   Source;
        BOOL    bFullDesktop;
        GUID    SessionType;
        GUID    LicenseType;
    } SESSIONENUM_LEVEL2,
        *PSESSIONENUM_LEVEL2;

    typedef struct _SESSIONENUM_LEVEL3 {
        LONG   SessionId;
        LONG   State;
        WCHAR  Name[33];
        ULONG  Source;
        BOOL   bFullDesktop;
        GUID   SessionType;
        GUID    LicenseType;
        ULONG  ProtoDataSize;
        [size_is(ProtoDataSize)]UCHAR *  pProtocolData;
    } SESSIONENUM_LEVEL3,
        *PSESSIONENUM_LEVEL3;

    typedef [switch_type(DWORD)] union _SessionInfo {
        [case(1)]
        SESSIONENUM_LEVEL1 SessionEnum_Level1;
        [case(2)]
        SESSIONENUM_LEVEL2 SessionEnum_Level2;
        [case(3)]
        SESSIONENUM_LEVEL3 SessionEnum_Level3;
        [default]
        ;
    } SessionInfo,
```

```
    *PSessionInfo;

typedef struct _SESSIONENUM {
    DWORD Level;
    [switch_is(Level)] SessionInfo Data;
} SESSIONENUM,
    *PSESSIONENUM;

HRESULT RpcOpenEnum(
    [in]        handle_t        hBinding,
    [out]       ENUM_HANDLE     *phEnum
    );

HRESULT RpcCloseEnum(
    [in,out]    ENUM_HANDLE     *phEnum
    );

HRESULT RpcFilterByState(
    [in]        ENUM_HANDLE     hEnum,
    [in]        LONG            State,
    [in]        BOOL            bInvert
    );

HRESULT RpcFilterByCallersName(
    [in]        ENUM_HANDLE     hEnum
    );

HRESULT RpcEnumAddFilter(
    [in]        ENUM_HANDLE     hEnum,
    [in]        ENUM_HANDLE     hSubEnum
    );

HRESULT RpcGetEnumResult(
    [in]        ENUM_HANDLE     hEnum,
    [out, size_is(,*pEntries)]
PSESSIONENUM   *ppSessionEnumResult,
    [in]        DWORD           Level,
    [out]       ULONG          *pEntries
    );

HRESULT RpcFilterBySessionType(
    [in] ENUM_HANDLE    hEnum,
    [in] GUID*          pSessionType
    );

HRESULT RpcFilterByLicenseType(
    [in] ENUM_HANDLE    hEnum,
    [in] GUID*          pLicenseType
    );

HRESULT RpcGetSessionIds(
    [in] handle_t                   hBinding,
    [in] SESSION_FILTER             Filter,
    [in, range( 0, 0xFFFF )] ULONG    MaxEntries,
    [out, size_is(,*pcSessionIds)]
LONG**                      pSessionIds,
    [out] ULONG*                    pcSessionIds
    );
```

```
}
```

6.2 rcmpublic.idl

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

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

//
//  public access ( local & remote )
//

[
    uuid(bde95fdf-eee0-45de-9e12-e5a61cd0d4fe),
    version(1.0),
    pointer default(unique)
]
interface RCMPublic
{
    static const WCHAR TSRCMRPC REMOTE ENDPOINT[] = L"\\pipe\\TermSrv API service";

    typedef struct {
        union switch (unsigned short sin family) u {
            case 2:
                struct {
                    USHORT sin_port;
                    ULONG  in addr;
                    UCHAR  sin zero[8];
                } ipv4;
            case 23:
                struct {
                    USHORT sin6 port;
                    ULONG  sin6 flowinfo;
                    USHORT sin6 addr[8];
                    ULONG  sin6 scope id;
                } ipv6;
        };
    } RCM REMOTEADDRESS, *PRCM REMOTEADDRESS;
    typedef WCHAR   LISTENER NAME[32];

#define LST ENUM LEVEL1          1
#define CURRENT_LST_ENUM_LEVEL   1

    typedef struct  LISTENERENUM LEVEL1 {
        LONG    Id;
        BOOL    bListening;
        WCHAR   Name[33];
    } LISTENERENUM_LEVEL1, *PLISTENERENUM_LEVEL1;

    typedef [switch type(DWORD)] union  ListenerInfo {
        [case(1)]
        LISTENERENUM LEVEL1 ListenerEnum Level1;
        [default]
        ;
    } ListenerInfo, *PListenerInfo;

    typedef struct  LISTENERENUM {
        DWORD Level;
        [switch_is(Level)] ListenerInfo Data;
```

```
} LISTENERENUM, *PLISTENERENUM;

HRESULT RpcGetClientData(
    [in]    handle t        hBinding,
    [in]    ULONG           SessionId,
    [out, size is( ,*pOutBuffByteLen )]
unsigned char **ppBuff,
    [out]   ULONG          *pOutBuffByteLen
    );

HRESULT RpcGetConfigData(
    [in]    handle t        hBinding,
    [in]    ULONG           SessionId,
    [out, size_is( ,*pOutBuffByteLen )]
unsigned char **ppBuff,
    [out]   ULONG  *pOutBuffByteLen
    );
typedef enum {
    PROTOCOLSTATUS INFO BASIC = 0,
    PROTOCOLSTATUS_INFO_EXTENDED = 1,
} PROTOCOLSTATUS INFO TYPE;

HRESULT RpcGetProtocolStatus(
    [in]    handle t        hBinding,
    [in]    ULONG           SessionId,
    [in]    PROTOCOLSTATUS_INFO_TYPE    InfoType,
    [out, size_is( ,*pcbProtoStatus )]
unsigned char **ppProtoStatus,
    [out]   ULONG           *pcbProtoStatus
    );

HRESULT RpcGetLastInputTime(
    [in]    handle_t        hBinding,
    [in]    ULONG           SessionId,
    [out]   hyper           *pLastInputTime
    );

HRESULT RpcGetRemoteAddress(
    [in]            handle_t        hBinding,
    [in]            ULONG           SessionId,
    [out]           PRCM REMOTEADDRESS pRemoteAddress
    );

HRESULT RpcShadow(
    [in]       handle t     hBinding,
    [in,unique,string]      WCHAR *szTargetServerName,
    [in]       ULONG        TargetSessionId,
    [in]       BYTE         HotKeyVk,
    [in]       USHORT       HotkeyModifiers
    );

HRESULT RpcShadowTarget(
    [in]                     handle t hBinding,
    [in]                     ULONG    SessionId,
    [in,size_is(ConfigSize)] PBYTE    pConfig,
    [in,range(0, 0x8000)]    ULONG    ConfigSize,
    [in,size is(AddressSize)]PBYTE    pAddress,
    [in,range(0, 0x1000)]    ULONG    AddressSize,
    [in,size is(ModuleDataSize)] PBYTE  pModuleData,
    [in,range(0, 0x1000)]    ULONG    ModuleDataSize,
    [in,size is(ThinwireDataSize)] PBYTE  pThinwireData,
    [in,range(0, 0x1000)]    ULONG    ThinwireDataSize,
    [in,string]              WCHAR    *szClientName
    );
```

```
    HRESULT RpcShadowStop(
        [in]        handle_t     hBinding,
        [in]        ULONG        SessionId
        );

    HRESULT RpcGetAllListeners(
        [in]        handle t        hBinding,
        [out, size_is(,*pNumListeners)] PLISTENERENUM *ppListeners,
        [in]        DWORD Level,
        [out]       ULONG   *pNumListeners
        );

    HRESULT RpcGetSessionProtocolLastInputTime(
        [in]    handle_t        hBinding,
        [in]    ULONG           SessionId,
        [in]    PROTOCOLSTATUS INFO TYPE    InfoType,
        [out, size is( ,*pcbProtoStatus )]
    unsigned char  **ppProtoStatus,
        [out]   ULONG           *pcbProtoStatus,
        [out]   hyper           *pLastInputTime
        );

    HRESULT RpcGetUserCertificates(
        [in] handle t hBinding,
        [in] ULONG SessionId,
        [out] ULONG* pcCerts,
        [out, size_is(, *pcbCerts)] BYTE** ppbCerts,
        [out] ULONG* pcbCerts
        );

    typedef enum {
        QUERY_SESSION_DATA_MODULE = 0,
        QUERY_SESSION_DATA_WDCONFIG,
        QUERY SESSION DATA VIRTUALDATA,
        QUERY SESSION DATA LICENSE,
        QUERY SESSION DATA DEVICEID
    } QUERY SESSION DATA TYPE;

    HRESULT RpcQuerySessionData(
        [in]       handle t                hBinding,
        [in]       ULONG                   SessionId,
        [in]       QUERY SESSION DATA TYPE type,
        [in, unique, size_is( cbInputData )] PBYTE  pbInputData,
        [in, range(0, 8192) ] DWORD        cbInputData,
        [out, ref, size is(cbSessionData), length is(*pcbReturnLength)]
    PBYTE                   pbSessionData,
        [in, range(0, 8192)] ULONG         cbSessionData,
        [out, ref] ULONG                  *pcbReturnLength,
        [out, ref] ULONG                  *pcbRequireBufferSize
        );
};

//
//  Describe a listener
//

[
    uuid(497d95a6-2d27-4bf5-9bbd-a6046957133c),
    version(1.0),
    pointer default(unique)
]
interface RCMListener
{
    typedef [context handle] void *HLISTENER;
    typedef [context_handle] void *HLISTENER_EXCLUSIVE;
```

```
    HRESULT RpcOpenListener(
        [in]            handle_t        hBinding,
        [in,string]     WCHAR           *szListenerName,
        [out]           HLISTENER       *phListener
        );

    HRESULT RpcCloseListener(
        [in,out]        HLISTENER_EXCLUSIVE *phListener
        );

    HRESULT RpcStopListener(
        [in]            HLISTENER       hListener
        );

    HRESULT RpcStartListener(
        [in]            HLISTENER       hListener
        );

    HRESULT RpcIsListening(
        [in]            HLISTENER       hListener,
        [out]           BOOL            *pbIsListening
        );
}
```

6.3 legacy.idl

For ease of implementation, the full IDL is provided, where "ms-dtyp.idl" is the IDL as specified in [MS-DTYP]Appendix A, "allproc.h" is as specified in section 6.7.

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

#include "allproc.h"

typedef UINT *        UINT_PTR;

static const WCHAR TSRPC_LEGACY_REMOTE_ENDPOINT[] =
L"\\pipe\\Ctx_WinStation_API_service";

typedef [context_handle] void *SERVER_HANDLE;
typedef [context_handle] void *SERVER_HANDLE_EXCLUSIVE;
typedef [context_handle] void *REGISTRATION_HANDLE;

[
    uuid(5ca4a760-ebb1-11cf-8611-00a0245420ed),
    version(1.0),
    pointer_default(unique)
]
interface IcaApi
{

    typedef [range(0, 0x8000)] ULONG BOUNDED_ULONG;

    BOOLEAN RpcWinStationOpenServer(
        [in]   handle_t      hBinding,
        [out]  DWORD         *pResult,
        [out]  SERVER_HANDLE *phServer
        );
```

```
BOOLEAN RpcWinStationCloseServer(
    [in]   SERVER_HANDLE hServer,
    [out]  DWORD        *pResult
    );

BOOLEAN RpcIcaServerPing(
    [in]   SERVER_HANDLE hServer,
    [out]  DWORD        *pResult
    );

BOOLEAN RpcWinStationEnumerate(
    [in]      SERVER_HANDLE  hServer,
    [out]     DWORD         *pResult,
    [in,out]  PULONG         pEntries,
    [in,out,unique,size_is(*pByteCount)]
PCHAR          pLogonId,
    [in,out]  PULONG         pByteCount,
    [in,out]  PULONG         pIndex
    );

BOOLEAN RpcWinStationRename(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in,size_is(NameOldSize)]
PWCHAR        pWinStationNameOld,
    [in,range(0, 256)]
DWORD         NameOldSize,
    [in,size_is(NameNewSize)]
PWCHAR pWinStationNameNew,
    [in,range(0, 256)]
DWORD         NameNewSize
    );

BOOLEAN RpcWinStationQueryInformation(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       DWORD         WinStationInformationClass,
    [in,out,unique,size_is(WinStationInformationLength)]
PCHAR         pWinStationInformation,
    [in, range( 0, 0x8000)]
DWORD         WinStationInformationLength,
    [out]      DWORD         *pReturnLength
    );

BOOLEAN RpcWinStationSetInformation(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       DWORD         WinStationInformationClass,
    [in,out,unique,size_is(WinStationInformationLength)]
PCHAR         pWinStationInformation,
    [in, range( 0, 0x8000)]
DWORD         WinStationInformationLength
    );

BOOLEAN RpcWinStationSendMessage(
```

```
    [in]       SERVER_HANDLE  hServer,
    [out]      DWORD          *pResult,
    [in]       DWORD          LogonId,
    [in,size_is(TitleLength)] PWCHAR pTitle,
    [in, range( 0, 1024 )]
DWORD          TitleLength,
    [in,size_is(MessageLength)]
PWCHAR         pMessage,
    [in, range( 0, 1024 )]
DWORD          MessageLength,
    [in]       DWORD          Style,
    [in]       DWORD          Timeout,
    [out]      DWORD          *pRespsonse,
    [in]       BOOLEAN        DoNotWait
    );

BOOLEAN RpcLogonIdFromWinStationName(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in,size_is(NameSize)]
PWCHAR        pWinStationName,
    [in, range( 0, 256 )]
DWORD         NameSize,
    [out]      DWORD         *pLogonId
    );

BOOLEAN RpcWinStationNameFromLogonId(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LoginId,
    [in,out,size_is(NameSize)]
PWCHAR        pWinStationName,
    [in, range( 0, 256 )]
DWORD         NameSize
    );

BOOLEAN RpcWinStationConnect(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         ClientLogonId,
    [in]       DWORD         ConnectLogonId,
    [in]       DWORD         TargetLogonId,
    [in,size_is(PasswordSize)] PWCHAR pPassword,
    [in, range(0, 1024)]
DWORD         PasswordSize,
    [in]       BOOLEAN       Wait
    );

BOOLEAN RpcWinStationVirtualOpen(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       DWORD         Pid,
    [in,size_is(NameSize)] PCHAR pVirtualName,
    [in,range(0, 1024)]
DWORD         NameSize,
    [out]      ULONG_PTR     *pHandle
    );
```

```
BOOLEAN RpcWinStationBeepOpen(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       DWORD         Pid,
    [out]      ULONG_PTR     *pHandle
    );

BOOLEAN RpcWinStationDisconnect(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       BOOLEAN       bWait
    );

BOOLEAN RpcWinStationReset(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       BOOLEAN       bWait
    );

BOOLEAN RpcWinStationShutdownSystem(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         ClientLogonId,
    [in]       DWORD         ShutdownFlags
    );

BOOLEAN RpcWinStationWaitSystemEvent(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         EventMask,
    [out]      DWORD         *pEventFlags
    );

BOOLEAN RpcWinStationShadow(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in,unique,size_is(NameSize)]
PWCHAR        pTargetServerName,
    [in, range( 0, 1024)]
DWORD         NameSize,
    [in]       DWORD         TargetLogonId,
    [in]       BYTE          HotKeyVk,
    [in]       USHORT        HotkeyModifiers
    );

BOOLEAN RpcWinStationShadowTargetSetup(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId
    );

BOOLEAN RpcWinStationShadowTarget(
    [in]       SERVER_HANDLE hServer,
```

```
    [out]       DWORD         *pResult,
    [in]        DWORD         LogonId,
    [in,size_is(ConfigSize)] PBYTE pConfig,
    [in, range( 0, 0x8000 )]
DWORD         ConfigSize,
    [in,size_is(AddressSize)] PBYTE pAddress,
    [in, range( 0, 0x1000 )]
DWORD         AddressSize,
    [in,size_is(ModuleDataSize)]
PBYTE         pModuleData,
    [in, range( 0, 0x1000 )]
DWORD         ModuleDataSize,
    [in,size_is(ThinwireDataSize)]
PBYTE         pThinwireData,
    [in]        DWORD         ThinwireDataSize,
    [in,size_is(ClientNameSize)]
PBYTE         pClientName,
    [in, range( 0, 1024 )]
DWORD         ClientNameSize
    );

VOID Opnum20NotUsedOnWire();

VOID Opnum21NotUsedOnWire();

VOID Opnum22NotUsedOnWire();

VOID Opnum23NotUsedOnWire();

VOID Opnum24NotUsedOnWire();

VOID Opnum25NotUsedOnWire();

BOOLEAN RpcWinStationSetPoolCount(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
    [in,out,size_is(LicenseSize)] PCHAR pLicense,
    [in, range( 0, 0x4000 )]
DWORD         LicenseSize
    );

BOOLEAN RpcWinStationQueryUpdateRequired(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
    [out]       DWORD         *pUpdateFlag
    );

BOOLEAN RpcWinStationCallback(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
    [in]        DWORD         LogonId,
    [in,size_is(PhoneNumberSize)] PWCHAR pPhoneNumber,
    [in, range( 0, 0x1000 )]
DWORD         PhoneNumberSize
    );

BOOLEAN RpcWinStationBreakPoint(
    [in]        SERVER_HANDLE hServer,
```

```
    [out]       DWORD         *pResult,
    [in]        DWORD         LogonId,
    [in]        BOOLEAN       KernelFlag
    );

BOOLEAN RpcWinStationReadRegistry(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult
    );

BOOLEAN RpcWinStationWaitForConnect(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
    [in]        DWORD         ClientLogonId,
    [in]        DWORD         ClientProcessId
    );

BOOLEAN RpcWinStationNotifyLogon(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
    [in]        DWORD         ClientLogonId,
    [in]        DWORD         ClientProcessId,
    [in]        BOOLEAN       fUserIsAdmin,
    [in]        DWORD         UserToken,
    [in,size_is(DomainSize)] PWCHAR pDomain,
    [in, range( 0, 1024 )]
DWORD         DomainSize,
    [in,size_is(UserNameSize)] PWCHAR pUserName,
    [in, range( 0, 1024 )]
DWORD         UserNameSize,
    [in,size_is(PasswordSize)] PWCHAR pPassword,
    [in, range( 0, 1024 )]
DWORD         PasswordSize,
    [in]         UCHAR         Seed,
    [in,size_is(ConfigSize)] PCHAR pUserConfig,
    [in, range( 0, 0x8000 )]
DWORD         ConfigSize,
    [out]        BOOLEAN       *pfIsRedirected
    );

BOOLEAN RpcWinStationNotifyLogoff(
    [in]        SERVER_HANDLE hServer,
    [in]        DWORD         ClientLogonId,
    [in]        DWORD         ClientProcessId,
    [out]       DWORD         *pResult
    );

VOID Opnum34NotUsedOnWire();

BOOLEAN RpcWinStationAnnoyancePopup(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
    [in]        DWORD         LogonIdld
    );

BOOLEAN RpcWinStationEnumerateProcesses(
    [in]        SERVER_HANDLE hServer,
    [out]       DWORD         *pResult,
```

```
    [out,size_is(ByteCount)] PBYTE pProcessBuffer,
    [in, range( 0, 0x8000 )]
DWORD         ByteCount
    );

BOOLEAN RpcWinStationTerminateProcess(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         ProcessId,
    [in]       DWORD         ExitCode
    );

void Opnum38NotUsedOnWire(void);

VOID Opnum39NotUsedOnWire();

VOID Opnum40NotUsedOnWire();

VOID Opnum41NotUsedOnWire();

BOOLEAN RpcWinStationNtsdDebug(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       LONG          ProcessId,
    [in]       ULONG         DbgProcessId,
    [in]       ULONG         DbgThreadId,
    [in]       DWORD_PTR     AttachCompletionRoutine
    );

BOOLEAN RpcWinStationGetAllProcesses(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       ULONG         Level,
    [in, out]  BOUNDED_ULONG         *pNumberOfProcesses,
    [out, size_is(,*pNumberOfProcesses)]
PTS_ALL_PROCESSES_INFO  *ppTsAllProcessesInfo
    );

BOOLEAN RpcWinStationGetProcessSid(
    [in]       SERVER_HANDLE hServer,
    [in]       DWORD         dwUniqueProcessId,
    [in]       LARGE_INTEGER ProcessStartTime,
    [out]      LONG          *pResult,
    [in,out,unique,size_is(dwSidSize)]
PBYTE         pProcessUserSid,
    [in, range( 0, 1024 )]
DWORD         dwSidSize,
    [in,out]   DWORD         *pdwSizeNeeded
    );

BOOLEAN RpcWinStationGetTermSrvCountersValue(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in, range( 0, 0x1000 )]
DWORD         dwEntries,
    [in,out, size_is(dwEntries)] PTS_COUNTER pCounter
    );
```

```
BOOLEAN RpcWinStationReInitializeSecurity(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult
    );

LONG RpcWinStationBroadcastSystemMessage(
    [in ]           SERVER_HANDLE hServer,
    [in]            ULONG       sessionID,
    [in]            ULONG       timeOut,
    [in]            DWORD       dwFlags,
    [in,out,ptr]    DWORD       *lpdwRecipients,
    [in]            ULONG       uiMessage,
    [in]            UINT_PTR    wParam,
    [in]            LONG_PTR    lParam,
    [in,size_is(bufferSize)] PBYTE pBuffer,
    [in, range( 0, 0x8000 )]
ULONG       bufferSize,
    [in]            BOOLEAN     fBufferHasValidData,
    [out]           LONG        *pResponse    );

LONG RpcWinStationSendWindowMessage(
    [in]           SERVER_HANDLE hServer,
    [in]           ULONG        sessionID,
    [in]           ULONG        timeOut,
    [in]           ULONG        hWnd,
    [in]           ULONG        Msg,
    [in]           UINT_PTR     wParam,
    [in]           LONG_PTR     lParam,
    [in,size_is(bufferSize)] PBYTE pBuffer,
    [in, range( 0, 0x8000 )]
ULONG        bufferSize,
    [in]           BOOLEAN      fBufferHasValidData,
    [out]          LONG         *pResponse    );

BOOLEAN
    RpcWinStationNotifyNewSession(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         ClientLogonId
    );

VOID Opnum50NotUsedOnWire();

VOID Opnum51NotUsedOnWire();

VOID Opnum52NotUsedOnWire();

BOOLEAN RpcWinStationGetLanAdapterName(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in, range( 0, 0x1000 )]
DWORD         PdNameSize,
    [in,size_is(PdNameSize)] PWCHAR pPdName,
    [in, range( 0, 1024) ]
ULONG         LanAdapter,
    [out]      ULONG         *pLength,
    [out, size_is(,*pLength)]
```

```
PWCHAR        *ppLanAdapter
    );

BOOLEAN RpcWinStationUpdateUserConfig(
    [in]       SERVER_HANDLE hServer,
    [in]       DWORD         ClientLogonId,
    [in]       DWORD         ClientProcessId,
    [in]       DWORD         UserToken,
    [out]      DWORD         *pResult
    );

BOOLEAN RpcWinStationQueryLogonCredentials(
    [in]       SERVER_HANDLE hServer,
    [in]       ULONG         LogonId,
    [out,size_is(,*pcbCredentials)]
PCHAR         *ppCredentials,
    [in,out]   ULONG         *pcbCredentials
    );

BOOLEAN RpcWinStationRegisterConsoleNotification(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       ULONG         SessionId,
    [in]       ULONG_PTR     hWnd,
    [in]       DWORD         dwFlags,
    [in]       DWORD         dwMask
    );

BOOLEAN RpcWinStationUnRegisterConsoleNotification(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       ULONG         SessionId,
    [in]       ULONG         hWnd
    );

BOOLEAN RpcWinStationUpdateSettings(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         SettingsClass,
    [in]       DWORD         SettingsParameters
    );

BOOLEAN RpcWinStationShadowStop(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         LogonId,
    [in]       BOOLEAN       bWait
    );

BOOLEAN RpcWinStationCloseServerEx(
    [in, out]  SERVER_HANDLE_EXCLUSIVE *phServer,
    [out]      DWORD        *pResult
    );

BOOLEAN RpcWinStationIsHelpAssistantSession(
    [in]    SERVER_HANDLE   hServer,
    [out]   DWORD*          pResult,
    [in]    ULONG           SessionId
```

```
    );

BOOLEAN RpcWinStationGetMachinePolicy(
    [in]           SERVER_HANDLE hServer,
    [in,out,size_is(bufferSize)]
PBYTE         pPolicy,
    [in, range( 0, 0x8000 )]
ULONG         bufferSize
    );

VOID Opnum63NotUsedOnWire();

VOID Opnum64NotUsedOnWire();

BOOLEAN RpcWinStationCheckLoopBack(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         ClientLogonId,
    [in]       DWORD         TargetLogonId,
    [in,size_is(NameSize)]
PWCHAR        pTargetServerName,
    [in, range( 0, 1024 )]
DWORD         NameSize
    );

BOOLEAN RpcConnectCallback(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       DWORD         TimeOut,
    [in]       ULONG         AddressType,
    [in,size_is(AddressSize)] PBYTE pAddress,
    [in, range( 0, 0x1000 )]
ULONG         AddressSize
    );

VOID Opnum67NotUsedOnWire();

VOID Opnum68NotUsedOnWire();

BOOLEAN RpcRemoteAssistancePrepareSystemRestore(
    [in]    SERVER_HANDLE   hServer,
    [out]   DWORD*          pResult
    );

BOOLEAN RpcWinStationGetAllProcesses_NT6(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [in]       ULONG         Level,
    [in, out]  BOUNDED_ULONG *pNumberOfProcesses,
    [out, size_is(,*pNumberOfProcesses)]
PTS_ALL_PROCESSES_INFO_NT6  *ppTsAllProcessesInfo
    );

BOOLEAN RpcWinStationRegisterNotificationEvent(
    [in]       SERVER_HANDLE hServer,
    [out]      DWORD         *pResult,
    [out]      REGISTRATION_HANDLE *pNotificationId,
    [in]       ULONG_PTR     EventHandle,
```

```
        [in]        DWORD          TargetSessionId,
        [in]        DWORD          dwMask,
        [in]        DWORD          dwProcessId
        );

    BOOLEAN RpcWinStationUnRegisterNotificationEvent(
        [in]        SERVER_HANDLE hServer,
        [out]       DWORD          *pResult,
        [in, out] REGISTRATION_HANDLE *NotificationId
        );

    BOOLEAN RpcWinStationAutoReconnect(
        [in]        SERVER_HANDLE hServer,
        [out]       DWORD          *pResult,
        [in]        DWORD          LogonId,
        [in]        DWORD          flags
        );

    BOOLEAN RpcWinStationCheckAccess(
        [in]        SERVER_HANDLE hServer,
        [out]       DWORD          *pResult,
        [in]        DWORD          ClientLogonId,
        [in]        DWORD          UserToken,
        [in]        ULONG          LogonId,
        [in]        ULONG          AccessMask
        );

    BOOLEAN RpcWinStationOpenSessionDirectory(
        [in]                        SERVER_HANDLE hServer,
        [out]       DWORD           *pResult,
        [in, string, max_is(64)]    PWCHAR pszServerName
        );
}
```

6.4 lcrpc.idl

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

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

typedef [context_handle] void *LCRPC_HANDLE;
typedef [context_handle] void *LCRPC_HANDLE_EXCLUSIVE;

/*
*  LCRPC Interface
*/

[
    uuid(2f59a331-bf7d-48cb-9e5c-7c090d76e8b8),
    version(1.0),
    pointer_default(unique),
    endpoint("ncalrpc:[LcRpc]")
]
```

```
interface LCRPC
{
    //
    //  RPC Open and Close Server Functions
    //

    BOOLEAN RpcLicensingOpenServer(
        [in] handle_t hBinding,
        [out] LCRPC_HANDLE *phServer,
        [out, ref] PLONG pStatus
        );

    void RpcLicensingCloseServer(
        [in, out] LCRPC_HANDLE_EXCLUSIVE *phServer
        );

    HRESULT Opnum2NotUsedOnWire(void);

    HRESULT Opnum3NotUsedOnWire(void);

    LONG RpcLicensingSetPolicy(
        [in] LCRPC_HANDLE hServer,
        [in] ULONG ulPolicyId,
        [out, ref] PLONG pNewPolicyStatus
        );

    //
    //  Administrative Functions
    //

    BOOLEAN RpcLicensingGetAvailablePolicyIds(
        [in] LCRPC_HANDLE hServer,
        [out, size_is(,*pcPolicies)] PULONG *ppulPolicyIds,
        [out, ref] PULONG pcPolicies,
        [out, ref] PLONG pStatus
        );

    BOOLEAN RpcLicensingGetPolicy(
        [in] LCRPC_HANDLE hServer,
        [out, ref] PULONG pulPolicyId,
        [out, ref] PLONG pStatus
        );

    BOOLEAN RpcLicensingGetPolicyInformation(
        [in] LCRPC_HANDLE hServer,
        [in] ULONG ulPolicyId,
        [in, out, ref] PULONG pulVersion,
        [out, size_is(,*pcbPolicyInfo)] PCHAR *ppPolicyInfo,
        [in, out, ref] PULONG pcbPolicyInfo,
        [out, ref] PLONG pStatus
        );

    HRESULT Opnum8NotUsedOnWire(void);

    BOOLEAN RpcLicensingServerPing(
        [in]  LCRPC_HANDLE hServer,
        [out, ref] PLONG pStatus
        );
```

```
    HRESULT RpcGetSessionUnderArbitration(
        [in] LCRPC_HANDLE hServer,
        [in] GUID* pTerminalType,
        [out, ref] PULONG pSessionId
        );
}
```

6.5 winsta.h

For ease of implementation, the full header is provided.

```
// [MS-TSTS] specific defines

#define WDPREFIX_LENGTH           12
#define STACK_ADDRESS_LENGTH     128
#define MAX_BR_NAME               65
#define DIRECTORY_LENGTH         256
#define INITIALPROGRAM_LENGTH    256
#define USERNAME_LENGTH           20
#define DOMAIN_LENGTH             17
#define PASSWORD_LENGTH           14
#define NASISPECIFICNAME_LENGTH   14
#define NASIUSERNAME_LENGTH       47
#define NASIPASSWORD_LENGTH       24
#define NASISESSIONNAME_LENGTH    16
#define NASIFILESERVER_LENGTH     47

#define CLIENTDATANAME_LENGTH      7
#define CLIENTNAME_LENGTH         20
#define CLIENTADDRESS_LENGTH      30
#define IMEFILENAME_LENGTH        32
#define DIRECTORY_LENGTH         256
#define CLIENTLICENSE_LENGTH      32
#define CLIENTMODEM_LENGTH        40
#define CLIENT_PRODUCT_ID_LENGTH  32
#define MAX_COUNTER_EXTENSIONS     2    /* actual value not known */
#define WINSTATIONNAME_LENGTH     32


typedef enum  SDCLASS {
    SdNone = 0,
    SdConsole,
    SdNetwork
} SDCLASS;

typedef enum  FLOWCONTROLCLASS {
    FlowControl None,
    FlowControl_Hardware,
    FlowControl_Software
} FLOWCONTROLCLASS;

typedef enum  WINSTATIONSTATECLASS {
    State Active = 0,
    State_Connected = 1,
    State_ConnectQuery = 2,
    State Shadow = 3,
    State Disconnected = 4,
    State Idle = 5,
    State_Listen = 6,
    State_Reset = 7,
```

```
    State_Down = 8,
    State_Init = 9
} WINSTATIONSTATECLASS;

typedef WCHAR   NASISPECIFICNAMEW[ NASISPECIFICNAME_LENGTH + 1 ];
typedef  CHAR   NASISPECIFICNAMEA[ NASISPECIFICNAME_LENGTH + 1 ];

typedef WCHAR   NASIUSERNAMEW[ NASIUSERNAME_LENGTH + 1 ];
typedef  CHAR   NASIUSERNAMEA[ NASIUSERNAME_LENGTH + 1 ];

typedef WCHAR   NASIPASSWORDW[ NASIPASSWORD_LENGTH + 1 ];
typedef  CHAR   NASIPASSWORDA[ NASIPASSWORD_LENGTH + 1 ];

typedef WCHAR   NASISESIONNAMEW[ NASISESSIONNAME_LENGTH + 1 ];
typedef  CHAR   NASISESIONNAMEA[ NASISESSIONNAME_LENGTH + 1 ];

typedef WCHAR   NASIFILESERVERW[ NASIFILESERVER_LENGTH + 1 ];
typedef  CHAR   NASIFILESERVERA[ NASIFILESERVER_LENGTH + 1 ];

typedef CHAR CLIENTDATANAME[ CLIENTDATANAME_LENGTH + 1 ];
typedef CHAR * PCLIENTDATANAME;

typedef WCHAR WINSTATIONNAMEW[ WINSTATIONNAME_LENGTH + 1 ];
typedef  CHAR WINSTATIONNAMEA[ WINSTATIONNAME_LENGTH + 1 ];

typedef struct _TS_SYSTEMTIME {
    USHORT wYear;
    USHORT wMonth;
    USHORT wDayOfWeek;
    USHORT wDay;
    USHORT wHour;
    USHORT wMinute;
    USHORT wSecond;
    USHORT wMilliseconds;
} TS_SYSTEMTIME;

typedef struct  TS_TIME_ZONE_INFORMATION {
    LONG Bias;
    WCHAR StandardName[32 ];
    TS_SYSTEMTIME StandardDate;
    LONG StandardBias;
    WCHAR DaylightName[32 ];
    TS_SYSTEMTIME DaylightDate;
    LONG DaylightBias;
} TS_TIME_ZONE_INFORMATION;


#ifdef  WIN64
typedef unsigned __int64 ULONG_PTR;
#else
typedef ULONG ULONG_PTR;
#endif

typedef ULONG_PTR SIZE_T;

typedef enum {
    SF_SERVICES_SESSION_POPUP
} SESSION_FILTER;

#define PROTOCOL_CONSOLE         0
#define PROTOCOL_ICA             1
#define PROTOCOL_TSHARE          2
#define PROTOCOL_RDP             2
#define PDNAME_LENGTH            32
#define WDNAME_LENGTH            32
```

```
#define CDNAME LENGTH            32
#define DEVICENAME_LENGTH        128
#define MODEMNAME_LENGTH         DEVICENAME_LENGTH
#define CALLBACK LENGTH          50
#define DLLNAME LENGTH           32
#define WINSTATIONCOMMENT LENGTH 60
#define MAX LICENSE SERVER LENGTH 1024
#define LOGONID_CURRENT     ((ULONG)-1)
#define MAX_PDCONFIG             10

#define TERMSRV TOTAL SESSIONS     1
#define TERMSRV DISC SESSIONS      2
#define TERMSRV RECON SESSIONS     3
#define TERMSRV_CURRENT_ACTIVE_SESSIONS 4
#define TERMSRV_CURRENT_DISC_SESSIONS   5
#define TERMSRV PENDING SESSIONS        6
#define TERMSRV SUCC TOTAL LOGONS       7
#define TERMSRV SUCC LOCAL LOGONS       8
#define TERMSRV SUCC REMOTE LOGONS      9
#define TERMSRV_SUCC_SESSION0_LOGONS   10
#define TERMSRV CURRENT TERMINATING SESSIONS  11
#define TERMSRV CURRENT LOGGEDON SESSIONS     12

#define NO FALLBACK DRIVERS  0x0
#define FALLBACK_BESTGUESS   0x1
#define FALLBACK_PCL         0x2
#define FALLBACK_PS          0x3
#define FALLBACK PCLANDPS    0x4

/*********************************
*   WinStationOpen access values
*********************************/

#define WINSTATION QUERY        0x00000001  /* WinStationQueryInformation() */
#define WINSTATION SET          0x00000002  /* WinStationSetInformation()   */
#define WINSTATION RESET        0x00000004  /* WinStationReset()            */
#define WINSTATION VIRTUAL      0x00000008  /* read/write direct data       */
#define WINSTATION_SHADOW       0x00000010  /* WinStationShadow()           */
#define WINSTATION_LOGON        0x00000020  /* logon to WinStation          */
#define WINSTATION LOGOFF       0x00000040  /* WinStationLogoff()           */
#define WINSTATION MSG          0x00000080  /* WinStationMsg()              */
#define WINSTATION CONNECT      0x00000100  /* WinStationConnect()          */
#define WINSTATION_DISCONNECT   0x00000200  /* WinStationDisconnect()       */
#define WINSTATION_GUEST_ACCESS (WINSTATION_LOGON)
#define WINSTATION CURRENT GUEST ACCESS (WINSTATION VIRTUAL | \
    WINSTATION LOGOFF)
#define WINSTATION USER ACCESS (WINSTATION GUEST ACCESS |    \
    WINSTATION QUERY |           \
    WINSTATION_CONNECT )
#define WINSTATION_CURRENT_USER_ACCESS (WINSTATION_SET |     \
    WINSTATION RESET |   \
    WINSTATION VIRTUAL | \
    WINSTATION LOGOFF |  \
    WINSTATION DISCONNECT)
#define WINSTATION_ALL_ACCESS ( STANDARD_RIGHTS_REQUIRED |   \
    WINSTATION_QUERY |           \
    WINSTATION SET |             \
    WINSTATION RESET |           \
    WINSTATION VIRTUAL |         \
    WINSTATION SHADOW |          \
    WINSTATION LOGON |           \
    WINSTATION_MSG |             \
    WINSTATION_CONNECT |         \
    WINSTATION DISCONNECT)
```

```
typedef WCHAR PDNAMEW[ PDNAME LENGTH + 1 ];
typedef WCHAR * PPDNAMEW;
typedef CHAR PDNAMEA[ PDNAME_LENGTH + 1 ];
typedef CHAR * PPDNAMEA;

#ifdef UNICODE
#define PDNAME PDNAMEW
#define PPDNAME PPDNAMEW
#else
#define PDNAME PDNAMEA
#define PPDNAME PPDNAMEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef WCHAR WDNAMEW[ WDNAME LENGTH + 1 ];
typedef WCHAR * PWDNAMEW;
typedef CHAR WDNAMEA[ WDNAME LENGTH + 1 ];
typedef CHAR * PWDNAMEA;
#ifdef UNICODE
#define WDNAME WDNAMEW
#define PWDNAME PWDNAMEW
#else
#define WDNAME WDNAMEA
#define PWDNAME PWDNAMEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef WCHAR CDNAMEW[ CDNAME LENGTH + 1 ];
typedef WCHAR * PCDNAMEW;
typedef CHAR CDNAMEA[ CDNAME_LENGTH + 1 ];
typedef CHAR * PCDNAMEA;
#ifdef UNICODE
#define CDNAME CDNAMEW
#define PCDNAME PCDNAMEW
#else
#define CDNAME CDNAMEA
#define PCDNAME PCDNAMEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef WCHAR DEVICENAMEW[ DEVICENAME_LENGTH + 1 ];
typedef WCHAR * PDEVICENAMEW;
typedef CHAR DEVICENAMEA[ DEVICENAME LENGTH + 1 ];
typedef CHAR * PDEVICENAMEA;
#ifdef UNICODE
#define DEVICENAME DEVICENAMEW
#define PDEVICENAME PDEVICENAMEW
#else
#define DEVICENAME DEVICENAMEA
#define PDEVICENAME PDEVICENAMEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef WCHAR MODEMNAMEW[ MODEMNAME LENGTH + 1 ];
typedef WCHAR * PMODEMNAMEW;
typedef CHAR MODEMNAMEA[ MODEMNAME LENGTH + 1 ];
typedef CHAR * PMODEMNAMEA;
#ifdef UNICODE
#define MODEMNAME MODEMNAMEW
#define PMODEMNAME PMODEMNAMEW
#else
```

```
#define MODEMNAME MODEMNAMEA
#define PMODEMNAME PMODEMNAMEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef WCHAR DLLNAMEW[ DLLNAME LENGTH + 1 ];
typedef WCHAR * PDLLNAMEW;
typedef CHAR DLLNAMEA[ DLLNAME_LENGTH + 1 ];
typedef CHAR * PDLLNAMEA;
#ifdef UNICODE
#define DLLNAME DLLNAMEW
#define PDLLNAME PDLLNAMEW
#else
#define DLLNAME DLLNAMEA
#define PDLLNAME PDLLNAMEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef WCHAR WDPREFIXW[ WDPREFIX LENGTH + 1 ];
typedef WCHAR * PWDPREFIXW;
typedef CHAR WDPREFIXA[ WDPREFIX LENGTH + 1 ];
typedef CHAR * PWDPREFIXA;

#ifdef UNICODE
#define WDPREFIX WDPREFIXW
#define PWDPREFIX PWDPREFIXW
#else
#define WDPREFIX WDPREFIXA
#define PWDPREFIX PWDPREFIXA
#endif /* UNICODE */

/*
*  Stack address structure
*/

typedef struct _ICA_STACK_ADDRESS {
    BYTE Address[ STACK_ADDRESS_LENGTH ];   // bytes 0,1 family, 2-n address
} ICA STACK ADDRESS, *PICA STACK ADDRESS;

typedef struct  ICA TRACE {
    WCHAR TraceFile[256];
    BOOLEAN fDebugger;
    BOOLEAN fTimestamp;
    ULONG TraceClass;
    ULONG TraceEnable;
    WCHAR TraceOption[64];
} ICA_TRACE, * PICA_TRACE;

#define EXTENDED USERNAME LEN 255
#define EXTENDED PASSWORD LEN 255
#define EXTENDED DOMAIN LEN 255

typedef struct _ExtendedClientCredentials {
    WCHAR UserName[EXTENDED_USERNAME_LEN + 1];
    WCHAR Password[EXTENDED PASSWORD LEN + 1];
    WCHAR Domain[EXTENDED DOMAIN LEN + 1] ;
}ExtendedClientCredentials, *pExtendedClientCredentials;

/*********************************
*  User Configuration structures
*********************************/

typedef WCHAR APPLICATIONNAMEW[ MAX_BR_NAME ];
```

```
typedef WCHAR *PAPPLICATIONNAMEW;
typedef CHAR APPLICATIONNAMEA[ MAX_BR_NAME ];
typedef CHAR *PAPPLICATIONNAMEA;
#ifdef UNICODE
#define APPLICATIONNAME APPLICATIONNAMEW
#define PAPPLICATIONNAME PAPPLICATIONNAMEW
#else
#define APPLICATIONNAME APPLICATIONNAMEA
#define PAPPLICATIONNAME PAPPLICATIONNAMEA
#endif /* UNICODE */

/*
*  Shadow options
*/

typedef enum  SHADOWCLASS {
    Shadow Disable,
    Shadow EnableInputNotify,
    Shadow EnableInputNoNotify,
    Shadow_EnableNoInputNotify,
    Shadow EnableNoInputNoNotify,
} SHADOWCLASS;

/*
*  Callback options
*/

typedef enum  CALLBACKCLASS {
    Callback Disable,
    Callback Roving,
    Callback Fixed,
} CALLBACKCLASS;

typedef struct  POLICY TS MACHINE
{
    ULONG   fPolicyDisableClip : 1 ;
    ULONG   fPolicyDisableCam : 1 ;
    ULONG   fPolicyDisableCcm : 1 ;
    ULONG   fPolicyDisableLPT : 1;
    ULONG   fPolicyDisableCpm : 1;
    ULONG   fPolicyPromptForPassword : 1 ;
    ULONG   fPolicyMaxInstanceCount : 1;
    ULONG   fPolicyMinEncryptionLevel : 1 ;
    ULONG   fPolicyFipsEnabled : 1;
    ULONG   fPolicyDisableAutoReconnect : 1;
    ULONG   fPolicyWFProfilePath: 1 ;
    ULONG   fPolicyWFHomeDir: 1 ;
    ULONG   fPolicyWFHomeDirDrive: 1 ;
    ULONG   fPolicyDenyTSConnections      : 1;
    ULONG   fPolicyTempFoldersPerSession  : 1;
    ULONG   fPolicyDeleteTempFoldersOnExit: 1;
    ULONG   fPolicyColorDepth  : 1;
    ULONG   fPolicySessionDirectoryActive  : 1;
    ULONG   fPolicySessionDirectoryLocation  : 1;
    ULONG   fPolicySessionDirectoryClusterName  : 1;
    ULONG   fPolicySessionDirectoryAdditionalParams  : 1;
    ULONG   fPolicySessionDirectoryExposeServerIP  : 1;
    ULONG   fPolicyPreventLicenseUpgrade  : 1;
    ULONG   fPolicySecureLicensing : 1;
    ULONG   fPolicyWritableTSCCPermissionsTAB : 1;
    ULONG   fPolicyDisableCdm : 1;
    ULONG   fPolicyForceClientLptDef : 1;
    ULONG   fPolicyShadow : 1 ;
    ULONG   fPolicyResetBroken : 1 ;
    ULONG   fPolicyReconnectSame : 1 ;
```

```
ULONG   fPolicyMaxSessionTime : 1 ;
ULONG   fPolicyMaxDisconnectionTime:1;
ULONG   fPolicyMaxIdleTime : 1 ;
ULONG   fPolicyInitialProgram : 1 ;
ULONG   fPolicySingleSessionPerUser : 1;
ULONG   fPolicyDisableWallpaper : 1;
ULONG   fPolicyKeepAlive   : 1;
ULONG   fPolicyEnableTimeZoneRedirection : 1;
ULONG   fPolicyDisableForcibleLogoff : 1;
ULONG   fPolicyLicensingMode : 1;
ULONG   fPolicyExplicitLSDiscovery: 1;
ULONG   fPolicyDisableTerminalServerTooltip:1;
ULONG   fDisableClip : 1 ;
ULONG   fDisableCam : 1 ;
ULONG   fDisableCcm : 1 ;
ULONG   fDisableLPT : 1;
ULONG   fDisableCpm : 1;
ULONG   fPromptForPassword : 1 ;
ULONG   ColorDepth : 3;
ULONG   fDenyTSConnections      : 1;
ULONG   fTempFoldersPerSession  : 1;
ULONG   fDeleteTempFoldersOnExit: 1;
ULONG   fWritableTSCCPermissionsTAB : 1;
ULONG   fDisableCdm  : 1;
ULONG   fForceClientLptDef : 1;
ULONG   fResetBroken : 1 ;
ULONG   fReconnectSame : 1 ;
ULONG   fSingleSessionPerUser:1;
ULONG   fDisableWallpaper : 1;
ULONG   fKeepAliveEnable : 1;
ULONG   fPreventLicenseUpgrade:1;
ULONG   fSecureLicensing:1;
ULONG   fEnableTimeZoneRedirection : 1;
ULONG   fDisableAutoReconnect : 1;
ULONG   fDisableForcibleLogoff : 1;
ULONG   fPolicyEncryptRPCTraffic : 1;
ULONG   fEncryptRPCTraffic : 1;
ULONG   fErrorInvalidProfile : 1;
ULONG   fPolicyFallbackPrintDriver : 1;
ULONG   FallbackPrintDriverType : 3;
ULONG   fDisableTerminalServerTooltip : 1;
BYTE    bSecurityLayer;
ULONG   fPolicySecurityLayer : 1;
BYTE    bUserAuthentication;
ULONG   fPolicyUserAuthentication : 1;
ULONG   fPolicyTurnOffSingleAppMode : 1;
ULONG   fTurnOffSingleAppMode : 1;
ULONG   fDisablePNPPolicyIsEnfored:1;
ULONG   fDisablePNPPolicyValue:1;
ULONG   MaxInstanceCount;
ULONG   LicensingMode;
BYTE    MinEncryptionLevel;
WCHAR   WFProfilePath[ DIRECTORY LENGTH + 1 ];
WCHAR   WFHomeDir[ DIRECTORY LENGTH + 1 ];
WCHAR   WFHomeDirDrive[ 4 ];
ULONG   SessionDirectoryActive;
WCHAR   SessionDirectoryLocation[DIRECTORY LENGTH+1];
WCHAR   SessionDirectoryClusterName[DIRECTORY LENGTH+1];
WCHAR   SessionDirectoryAdditionalParams[DIRECTORY LENGTH+1];
ULONG   SessionDirectoryExposeServerIP;
ULONG   KeepAliveInterval;
SHADOWCLASS     Shadow;
ULONG           MaxConnectionTime;
ULONG           MaxDisconnectionTime;
ULONG           MaxIdleTime;
```

```
    WCHAR WorkDirectory[ DIRECTORY LENGTH + 1 ];
    WCHAR InitialProgram[ INITIALPROGRAM_LENGTH + 1 ];
    WCHAR LicenseServers[MAX_LICENSE_SERVER_LENGTH + 1 ];
} POLICY TS MACHINE, *PPOLICY TS MACHINE;

/*
*  User Configuration data
*/

typedef struct  USERCONFIGW {
    /* if flag is set inherit parameter from user or client configuration */
    ULONG fInheritAutoLogon : 1;
    ULONG fInheritResetBroken : 1;
    ULONG fInheritReconnectSame : 1;
    ULONG fInheritInitialProgram : 1;
    ULONG fInheritCallback : 1;
    ULONG fInheritCallbackNumber : 1;
    ULONG fInheritShadow : 1;
    ULONG fInheritMaxSessionTime : 1;
    ULONG fInheritMaxDisconnectionTime : 1;
    ULONG fInheritMaxIdleTime : 1;
    ULONG fInheritAutoClient : 1;
    ULONG fInheritSecurity : 1;
    ULONG fPromptForPassword : 1;
    ULONG fResetBroken : 1;
    ULONG fReconnectSame : 1;
    ULONG fLogonDisabled : 1;
    ULONG fWallPaperDisabled : 1;
    ULONG fAutoClientDrives : 1;
    ULONG fAutoClientLpts : 1;
    ULONG fForceClientLptDef : 1;
    ULONG fRequireEncryption : 1;
    ULONG fDisableEncryption : 1;
    ULONG fUnused1 : 1;
    ULONG fHomeDirectoryMapRoot : 1;
    ULONG fUseDefaultGina : 1;
    ULONG fCursorBlinkDisabled : 1;
    ULONG fPublishedApp : 1;
    ULONG fHideTitleBar : 1;
    ULONG fMaximize : 1;
    ULONG fDisableCpm : 1;
    ULONG fDisableCdm : 1;
    ULONG fDisableCcm : 1;
    ULONG fDisableLPT : 1;
    ULONG fDisableClip : 1;
    ULONG fDisableExe : 1;
    ULONG fDisableCam : 1;
    ULONG fDisableAutoReconnect : 1;
    ULONG ColorDepth : 3;
    ULONG fInheritColorDepth: 1;
    ULONG   fErrorInvalidProfile : 1;
    ULONG fPasswordIsScPin: 1;
    ULONG   fDisablePNPRedir:1;
    WCHAR UserName[ USERNAME LENGTH + 1 ];
    WCHAR Domain[ DOMAIN_LENGTH + 1 ];
    WCHAR Password[ PASSWORD_LENGTH + 1 ];
    WCHAR WorkDirectory[ DIRECTORY LENGTH + 1 ];
    WCHAR InitialProgram[ INITIALPROGRAM LENGTH + 1 ];
    WCHAR CallbackNumber[ CALLBACK LENGTH + 1 ];
    CALLBACKCLASS Callback;
    SHADOWCLASS Shadow;
    ULONG MaxConnectionTime;
    ULONG MaxDisconnectionTime;
    ULONG MaxIdleTime;
    ULONG KeyboardLayout;
```

```
    BYTE MinEncryptionLevel;
    WCHAR NWLogonServer[ NASIFILESERVER_LENGTH + 1 ];
    APPLICATIONNAMEW PublishedName;
    WCHAR WFProfilePath[ DIRECTORY LENGTH + 1 ];
    WCHAR WFHomeDir[ DIRECTORY LENGTH + 1 ];
    WCHAR WFHomeDirDrive[ 4 ];
} USERCONFIGW, * PUSERCONFIGW;

typedef struct _USERCONFIGA {
    ULONG fInheritAutoLogon : 1;
    ULONG fInheritResetBroken : 1;
    ULONG fInheritReconnectSame : 1;
    ULONG fInheritInitialProgram : 1;
    ULONG fInheritCallback : 1;
    ULONG fInheritCallbackNumber : 1;
    ULONG fInheritShadow : 1;
    ULONG fInheritMaxSessionTime : 1;
    ULONG fInheritMaxDisconnectionTime : 1;
    ULONG fInheritMaxIdleTime : 1;
    ULONG fInheritAutoClient : 1;
    ULONG fInheritSecurity : 1;
    ULONG fPromptForPassword : 1;
    ULONG fResetBroken : 1;
    ULONG fReconnectSame : 1;
    ULONG fLogonDisabled : 1;
    ULONG fWallPaperDisabled : 1;
    ULONG fAutoClientDrives : 1;
    ULONG fAutoClientLpts : 1;
    ULONG fForceClientLptDef : 1;
    ULONG fRequireEncryption : 1;
    ULONG fDisableEncryption : 1;
    ULONG fUnused1 : 1;
    ULONG fHomeDirectoryMapRoot : 1;
    ULONG fUseDefaultGina : 1;
    ULONG fCursorBlinkDisabled : 1;
    ULONG fPublishedApp : 1;
    ULONG fHideTitleBar : 1;
    ULONG fMaximize : 1;
    ULONG fDisableCpm : 1;
    ULONG fDisableCdm : 1;
    ULONG fDisableCcm : 1;
    ULONG fDisableLPT : 1;
    ULONG fDisableClip : 1;
    ULONG fDisableExe : 1;
    ULONG fDisableCam : 1;
    ULONG fDisableAutoReconnect : 1;
    ULONG   fErrorInvalidProfile : 1;
    ULONG fPasswordIsScPin: 1;
    ULONG   fDisablePNPRedir:1;
    CHAR UserName[ USERNAME_LENGTH + 1 ];
    CHAR Domain[ DOMAIN LENGTH + 1 ];
    CHAR Password[ PASSWORD LENGTH + 1 ];
    CHAR WorkDirectory[ DIRECTORY LENGTH + 1 ];
    CHAR InitialProgram[ INITIALPROGRAM LENGTH + 1 ];
    CHAR CallbackNumber[ CALLBACK_LENGTH + 1 ];
    CALLBACKCLASS Callback;
    SHADOWCLASS Shadow;
    ULONG MaxConnectionTime;
    ULONG MaxDisconnectionTime;
    ULONG MaxIdleTime;
    ULONG KeyboardLayout;
    BYTE MinEncryptionLevel;
    CHAR NWLogonServer[ NASIFILESERVER_LENGTH + 1 ];
    APPLICATIONNAMEA PublishedName;
    CHAR WFProfilePath[ DIRECTORY_LENGTH + 1 ];
```

```
    CHAR WFHomeDir[ DIRECTORY LENGTH + 1 ];
    CHAR WFHomeDirDrive[ 4 ];
} USERCONFIGA, * PUSERCONFIGA;

#ifdef UNICODE
#define USERCONFIG USERCONFIGW
#define PUSERCONFIG PUSERCONFIGW
#else
#define USERCONFIG USERCONFIGA
#define PUSERCONFIG PUSERCONFIGA
#endif /* UNICODE */

/******************
*  PD structures
******************/

typedef struct  PDCONFIG2W{
    PDNAMEW PdName;
    SDCLASS SdClass;
    DLLNAMEW PdDLL;
    ULONG    PdFlag;
    ULONG OutBufLength;
    ULONG OutBufCount;
    ULONG OutBufDelay;
    ULONG InteractiveDelay;
    ULONG PortNumber;
    ULONG KeepAliveTimeout;
} PDCONFIG2W, * PPDCONFIG2W;

typedef struct  PDCONFIG2A {
    PDNAMEA PdName;
    SDCLASS SdClass;
    DLLNAMEA PdDLL;
    ULONG    PdFlag;
    ULONG OutBufLength;
    ULONG OutBufCount;
    ULONG OutBufDelay;
    ULONG InteractiveDelay;
    ULONG PortNumber;
    ULONG KeepAliveTimeout;
} PDCONFIG2A, * PPDCONFIG2A;

/*
*  PdFlag defines
*/

#define PD UNUSED      0x00000001
#define PD RELIABLE    0x00000002
#define PD_FRAME       0x00000004
#define PD_CONNECTION  0x00000008
#define PD CONSOLE     0x00000010
#define PD LANA        0x00000020
#define PD TRANSPORT   0x00000040
#define PD SINGLE INST 0x00000080
#define PD_NOLOW_WATERMARK 0x00000100
#ifdef UNICODE
#define PDCONFIG2 PDCONFIG2W
#define PPDCONFIG2 PPDCONFIG2W
#else
#define PDCONFIG2 PDCONFIG2A
#define PPDCONFIG2 PPDCONFIG2A
#endif /* UNICODE */
/*-----------------------------------------------*/
typedef enum  RECEIVEFLOWCONTROLCLASS {
    ReceiveFlowControl_None,
```

```
    ReceiveFlowControl RTS,
    ReceiveFlowControl_DTR,
} RECEIVEFLOWCONTROLCLASS;

typedef enum  TRANSMITFLOWCONTROLCLASS {
    TransmitFlowControl None,
    TransmitFlowControl CTS,
    TransmitFlowControl_DSR,
} TRANSMITFLOWCONTROLCLASS;

typedef struct  FLOWCONTROLCONFIG {
    ULONG fEnableSoftwareTx: 1;
    ULONG fEnableSoftwareRx: 1;
    ULONG fEnableDTR: 1;
    ULONG fEnableRTS: 1;
    CHAR XonChar;
    CHAR XoffChar;
    FLOWCONTROLCLASS Type;
    RECEIVEFLOWCONTROLCLASS HardwareReceive;
    TRANSMITFLOWCONTROLCLASS HardwareTransmit;
} FLOWCONTROLCONFIG, * PFLOWCONTROLCONFIG;

typedef enum  ASYNCCONNECTCLASS {
    Connect CTS,
    Connect_DSR,
    Connect_RI,
    Connect_DCD,
    Connect FirstChar,
    Connect Perm,
} ASYNCCONNECTCLASS;

typedef struct _CONNECTCONFIG {
    ASYNCCONNECTCLASS Type;
    ULONG fEnableBreakDisconnect: 1;
} CONNECTCONFIG, * PCONNECTCONFIG;
/*-------------------------------------------------*/

typedef struct _ASYNCCONFIGW {
    DEVICENAMEW DeviceName;
    MODEMNAMEW ModemName;
    ULONG BaudRate;
    ULONG Parity;
    ULONG StopBits;
    ULONG ByteSize;
    ULONG fEnableDsrSensitivity: 1;
    ULONG fConnectionDriver: 1;
    FLOWCONTROLCONFIG FlowControl;
    CONNECTCONFIG Connect;
} ASYNCCONFIGW, * PASYNCCONFIGW;

typedef struct  ASYNCCONFIGA {
    DEVICENAMEA DeviceName;
    MODEMNAMEA ModemName;
    ULONG BaudRate;
    ULONG Parity;
    ULONG StopBits;
    ULONG ByteSize;
    ULONG fEnableDsrSensitivity: 1;
    ULONG fConnectionDriver: 1;
    FLOWCONTROLCONFIG FlowControl;
    CONNECTCONFIG Connect;
} ASYNCCONFIGA, * PASYNCCONFIGA;

#ifdef UNICODE
#define ASYNCCONFIG ASYNCCONFIGW
```

```
#define PASYNCCONFIG PASYNCCONFIGW
#else
#define ASYNCCONFIG ASYNCCONFIGA
#define PASYNCCONFIG PASYNCCONFIGA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef struct _NETWORKCONFIGW {
    LONG LanAdapter;
    DEVICENAMEW NetworkName;
    ULONG Flags;
} NETWORKCONFIGW, * PNETWORKCONFIGW;

typedef struct _NETWORKCONFIGA {
    LONG LanAdapter;
    DEVICENAMEA NetworkName;
    ULONG Flags;
} NETWORKCONFIGA, * PNETWORKCONFIGA;

#define NETWORK CLIENT  0x00000001
#ifdef UNICODE
#define NETWORKCONFIG NETWORKCONFIGW
#define PNETWORKCONFIG PNETWORKCONFIGW
#else
#define NETWORKCONFIG NETWORKCONFIGA
#define PNETWORKCONFIG PNETWORKCONFIGA
#endif /* UNICODE */
/*-------------------------------------------------*/
#ifdef UNICODE
#define NASICONFIG NASICONFIGW
#define PNASICONFIG PNASICONFIGW
#else
#define NASICONFIG NASICONFIGA
#define PNASICONFIG PNASICONFIGA
#endif /* UNICODE */

typedef struct _NASICONFIGW {
    NASISPECIFICNAMEW    SpecificName;
    NASIUSERNAMEW        UserName;
    NASIPASSWORDW        PassWord;
    NASISESIONNAMEW      SessionName;
    NASIFILESERVERW      FileServer;
    BOOLEAN              GlobalSession;
} NASICONFIGW, * PNASICONFIGW;

typedef struct  NASICONFIGA {
    NASISPECIFICNAMEA    SpecificName;
    NASIUSERNAMEA        UserName;
    NASIPASSWORDA        PassWord;
    NASISESIONNAMEA      SessionName;
    NASIFILESERVERA      FileServer;
    BOOLEAN              GlobalSession;
} NASICONFIGA, * PNASICONFIGA;

/*-------------------------------------------------*/

typedef struct  OEMTDCONFIGW {
    LONG Adapter;
    DEVICENAMEW DeviceName;
    ULONG Flags;
} OEMTDCONFIGW, * POEMTDCONFIGW;

typedef struct  OEMTDCONFIGA {
    LONG Adapter;
```

```
    DEVICENAMEA DeviceName;
    ULONG Flags;
} OEMTDCONFIGA, * POEMTDCONFIGA;

#ifdef UNICODE
#define OEMTDCONFIG OEMTDCONFIGW
#define POEMTDCONFIG POEMTDCONFIGW
#else
#define OEMTDCONFIG OEMTDCONFIGA
#define POEMTDCONFIG POEMTDCONFIGA
#endif /* UNICODE */

/*-----------------------------------------------*/

typedef struct _PDPARAMSW {
    SDCLASS SdClass;
    union {
        NETWORKCONFIGW Network;
        ASYNCCONFIGW Async;
        NASICONFIGW Nasi;
        OEMTDCONFIGW OemTd;
    };
} PDPARAMSW, * PPDPARAMSW;

typedef struct _PDPARAMSA {
    SDCLASS SdClass;
    union {
        NETWORKCONFIGA Network;
        ASYNCCONFIGA Async;
        NASICONFIGA Nasi;
        OEMTDCONFIGA OemTd;
    };
} PDPARAMSA, * PPDPARAMSA;

#ifdef UNICODE
#define PDPARAMS PDPARAMSW
#define PPDPARAMS PPDPARAMSW
#else
#define PDPARAMS PDPARAMSA
#define PPDPARAMS PPDPARAMSA
#endif /* UNICODE */

/*-----------------------------------------------*/

typedef struct  PDCONFIGW {
    PDCONFIG2W Create;
    PDPARAMSW Params;
} PDCONFIGW, * PPDCONFIGW;

typedef struct _PDCONFIGA {
    PDCONFIG2A Create;
    PDPARAMSA Params;
} PDCONFIGA, * PPDCONFIGA;

#ifdef UNICODE
#define PDCONFIG PDCONFIGW
#define PPDCONFIG PPDCONFIGW
#else
#define PDCONFIG PDCONFIGA
#define PPDCONFIG PPDCONFIGA
#endif /* UNICODE */

/***********************
*  Wd structures
***********************/
```

```
typedef struct _WDCONFIGW {
    WDNAMEW WdName;
    DLLNAMEW WdDLL;
    DLLNAMEW WsxDLL;
    ULONG WdFlag;
    ULONG WdInputBufferLength;
    DLLNAMEW CfgDLL;
    WDPREFIXW WdPrefix;
} WDCONFIGW, * PWDCONFIGW;

typedef struct  WDCONFIGA {
    WDNAMEA WdName;
    DLLNAMEA WdDLL;
    DLLNAMEA WsxDLL;
    ULONG WdFlag;
    ULONG WdInputBufferLength;
    DLLNAMEA CfgDLL;
    WDPREFIXA WdPrefix;
} WDCONFIGA, * PWDCONFIGA;

/*
*  WdFlag defines
*/

#define WDF_UNUSED            0x00000001
#define WDF_SHADOW_SOURCE     0x00000002
#define WDF SHADOW TARGET     0x00000004
#define WDF ICA               0x00000008
#define WDF TSHARE            0x00000010
#define WDF DYNAMIC RECONNECT 0x00000020
#define WDF_USER_VCIOCTL      0x00000040
#define WDF_SUBDESKTOP        0x00008000
#ifdef UNICODE
#define WDCONFIG WDCONFIGW
#define PWDCONFIG PWDCONFIGW
#else
#define WDCONFIG WDCONFIGA
#define PWDCONFIG PWDCONFIGA
#endif /* UNICODE */

/************************************
*  Connection Driver structures (CD)
************************************/

/*
*  connection driver classes
*/

typedef enum _CDCLASS {
    CdNone,
    CdModem,
    CdClass Maximum,
} CDCLASS;

/*-----------------------------------------------*/

typedef struct  CDCONFIGW {
    CDCLASS CdClass;
    CDNAMEW CdName;
    DLLNAMEW CdDLL;
    ULONG CdFlag;
} CDCONFIGW, * PCDCONFIGW;

typedef struct _CDCONFIGA {
```

```
    CDCLASS CdClass;
    CDNAMEA CdName;
    DLLNAMEA CdDLL;
    ULONG CdFlag;
} CDCONFIGA, * PCDCONFIGA;

#ifdef UNICODE
#define CDCONFIG CDCONFIGW
#define PCDCONFIG PCDCONFIGW
#else
#define CDCONFIG CDCONFIGA
#define PCDCONFIG PCDCONFIGA
#endif /* UNICODE */

/*****************************
*  Window Station structures
*****************************/

typedef struct  WINSTATIONCREATEW {
    ULONG fEnableWinStation : 1;
    ULONG MaxInstanceCount;
} WINSTATIONCREATEW, * PWINSTATIONCREATEW;

typedef struct  WINSTATIONCREATEA {
    ULONG fEnableWinStation : 1;
    ULONG MaxInstanceCount;
} WINSTATIONCREATEA, * PWINSTATIONCREATEA;

#ifdef UNICODE
#define WINSTATIONCREATE WINSTATIONCREATEW
#define PWINSTATIONCREATE PWINSTATIONCREATEW
#else
#define WINSTATIONCREATE WINSTATIONCREATEA
#define PWINSTATIONCREATE PWINSTATIONCREATEA
#endif /* UNICODE */

/*-------------------------------------------------*/

typedef struct _WINSTATIONCONFIGW {
    WCHAR Comment[ WINSTATIONCOMMENT LENGTH + 1 ];
    USERCONFIGW User;
    char OEMId[4];
} WINSTATIONCONFIGW, * PWINSTATIONCONFIGW;

typedef struct  WINSTATIONCONFIGA {
    CHAR Comment[ WINSTATIONCOMMENT LENGTH + 1 ];
    USERCONFIGA User;
    char OEMId[4];
} WINSTATIONCONFIGA, * PWINSTATIONCONFIGA;

#ifdef UNICODE
#define WINSTATIONCONFIG WINSTATIONCONFIGW
#define PWINSTATIONCONFIG PWINSTATIONCONFIGW
#else
#define WINSTATIONCONFIG WINSTATIONCONFIGA
#define PWINSTATIONCONFIG PWINSTATIONCONFIGA
#endif /* UNICODE */

/*-------------------------------------------------*/

#define EXECSRVPIPENAMELEN 48

typedef enum _WINSTATIONINFOCLASS {
    WinStationCreateData,
    WinStationConfiguration,
```

```
    WinStationPdParams,
    WinStationWd,
    WinStationPd,
    WinStationPrinter,
    WinStationClient,
    WinStationModules,
    WinStationInformation,
    WinStationTrace,
    WinStationBeep,
    WinStationEncryptionOff,
    WinStationEncryptionPerm,
    WinStationNtSecurity,
    WinStationUserToken,
    WinStationUnused1,
    WinStationVideoData,
    WinStationInitialProgram,
    WinStationCd,
    WinStationSystemTrace,
    WinStationVirtualData,
    WinStationClientData,
    WinStationSecureDesktopEnter,
    WinStationSecureDesktopExit,
    WinStationLoadBalanceSessionTarget,
    WinStationLoadIndicator,
    WinStationShadowInfo,
    WinStationDigProductId,
    WinStationLockedState,
    WinStationRemoteAddress,
    WinStationIdleTime,
    WinStationLastReconnectType,
    WinStationDisallowAutoReconnect,
    WinStationMprNotifyInfo,
    WinStationExecSrvSystemPipe,
    WinStationSmartCardAutoLogon,
    WinStationIsAdminLoggedOn,
    WinStationReconnectedFromId,
    WinStationEffectsPolicy
} WINSTATIONINFOCLASS;

/*-----------------------------------------------*/

typedef struct  WINSTATIONCLIENTDATA {
    CLIENTDATANAME DataName;
    BOOLEAN fUnicodeData;
    /* BYTE   Data[1]; Variable length data follows */
} WINSTATIONCLIENTDATA, * PWINSTATIONCLIENTDATA;

/*-----------------------------------------------*/

typedef struct _WINSTATIONUSERTOKEN {
    HANDLE ProcessId;
    HANDLE ThreadId;
    HANDLE UserToken;
} WINSTATIONUSERTOKEN, * PWINSTATIONUSERTOKEN;

/*-----------------------------------------------*/

typedef struct  WINSTATIONVIDEODATA {
    USHORT  HResolution;
    USHORT  VResolution;
    USHORT  fColorDepth;
} WINSTATIONVIDEODATA, *PWINSTATIONVIDEODATA;

/*-----------------------------------------------*/
```

```
typedef struct  WINSTATIONCONFIG2W {
    WINSTATIONCREATEW Create;
    PDCONFIGW Pd[ MAX_PDCONFIG ];
    WDCONFIGW Wd;
    CDCONFIGW Cd;
    WINSTATIONCONFIGW   Config;
} WINSTATIONCONFIG2W, * PWINSTATIONCONFIG2W;

typedef struct _WINSTATIONCONFIG2A {
    WINSTATIONCREATEA Create;
    PDCONFIGA Pd[ MAX PDCONFIG ];
    WDCONFIGA Wd;
    CDCONFIGA Cd;
    WINSTATIONCONFIGA Config;
} WINSTATIONCONFIG2A, * PWINSTATIONCONFIG2A;

#ifdef UNICODE
#define WINSTATIONCONFIG2 WINSTATIONCONFIG2W
#define PWINSTATIONCONFIG2 PWINSTATIONCONFIG2W
#else
#define WINSTATIONCONFIG2 WINSTATIONCONFIG2A
#define PWINSTATIONCONFIG2 PWINSTATIONCONFIG2A
#endif /* UNICODE */

/*
*  WinStation client data structure
*/

typedef struct  WINSTATIONCLIENTW {
    ULONG fTextOnly: 1;
    ULONG fDisableCtrlAltDel: 1;
    ULONG fMouse: 1;
    ULONG fDoubleClickDetect: 1;
    ULONG fINetClient: 1;
    ULONG fPromptForPassword : 1;
    ULONG fMaximizeShell: 1;
    ULONG fEnableWindowsKey: 1;
    ULONG fRemoteConsoleAudio: 1;
    ULONG fPasswordIsScPin: 1;
    ULONG fNoAudioPlayback: 1;
    ULONG fUsingSavedCreds: 1;
    WCHAR ClientName[ CLIENTNAME LENGTH + 1 ];
    WCHAR Domain[ DOMAIN_LENGTH + 1 ];
    WCHAR UserName[ USERNAME_LENGTH + 1 ];
    WCHAR Password[ PASSWORD LENGTH + 1 ];
    WCHAR WorkDirectory[ DIRECTORY LENGTH + 1 ];
    WCHAR InitialProgram[ INITIALPROGRAM LENGTH + 1 ];
    ULONG SerialNumber;
    BYTE EncryptionLevel;
    ULONG ClientAddressFamily;
    WCHAR ClientAddress[ CLIENTADDRESS LENGTH + 1 ];
    USHORT HRes;
    USHORT VRes;
    USHORT ColorDepth;
    USHORT ProtocolType;
    ULONG KeyboardLayout;
    ULONG KeyboardType;
    ULONG KeyboardSubType;
    ULONG KeyboardFunctionKey;
    WCHAR imeFileName[ IMEFILENAME LENGTH + 1 ];
    WCHAR ClientDirectory[ DIRECTORY LENGTH + 1 ];
    WCHAR ClientLicense[ CLIENTLICENSE_LENGTH + 1 ];
    WCHAR ClientModem[ CLIENTMODEM_LENGTH + 1 ];
    ULONG ClientBuildNumber;
    ULONG ClientHardwareId;
```

```
    USHORT ClientProductId;
    USHORT OutBufCountHost;
    USHORT OutBufCountClient;
    USHORT OutBufLength;
    WCHAR AudioDriverName[9];
    TS TIME ZONE INFORMATION ClientTimeZone;
    ULONG ClientSessionId;
    WCHAR clientDigProductId[CLIENT_PRODUCT_ID_LENGTH];
    ULONG PerformanceFlags;
    ULONG ActiveInputLocale;
} WINSTATIONCLIENTW, * PWINSTATIONCLIENTW;

/*
*  WinStation client data structure
*/

typedef struct  WINSTATIONCLIENTA {
    ULONG fTextOnly: 1;
    ULONG fDisableCtrlAltDel: 1;
    ULONG fMouse: 1;
    ULONG fDoubleClickDetect: 1;
    ULONG fINetClient: 1;
    ULONG fPromptForPassword : 1;
    ULONG fMaximizeShell: 1;
    ULONG fEnableWindowsKey: 1;
    ULONG fRemoteConsoleAudio: 1;
    ULONG fPasswordIsScPin: 1;
    ULONG fNoAudioPlayback: 1;
    ULONG fUsingSavedCreds: 1;
    char ClientName[ CLIENTNAME LENGTH + 1 ];
    char Domain[ DOMAIN LENGTH + 1 ];
    char UserName[ USERNAME_LENGTH + 1 ];
    char Password[ PASSWORD_LENGTH + 1 ];
    char WorkDirectory[ DIRECTORY LENGTH + 1 ];
    char InitialProgram[ INITIALPROGRAM LENGTH + 1 ];
    ULONG SerialNumber;
    BYTE EncryptionLevel;
    ULONG ClientAddressFamily;
    char ClientAddress[ CLIENTADDRESS_LENGTH + 1 ];
    USHORT HRes;
    USHORT VRes;
    USHORT ColorDepth;
    USHORT ProtocolType;
    ULONG KeyboardLayout;
    ULONG KeyboardType;
    ULONG KeyboardSubType;
    ULONG KeyboardFunctionKey;
    char imeFileName[ IMEFILENAME LENGTH + 1 ];
    char ClientDirectory[ DIRECTORY_LENGTH + 1 ];
    char ClientLicense[ CLIENTLICENSE_LENGTH + 1 ];
    char ClientModem[ CLIENTMODEM LENGTH + 1 ];
    ULONG ClientBuildNumber;
    ULONG ClientHardwareId;
    USHORT ClientProductId;
    USHORT OutBufCountHost;
    USHORT OutBufCountClient;
    USHORT OutBufLength;
    char AudioDriverName[9];
    TS TIME ZONE INFORMATION ClientTimeZone;
    ULONG ClientSessionId;
    char clientDigProductId[CLIENT PRODUCT ID LENGTH];
    ULONG PerformanceFlags;
    ULONG ActiveInputLocale;
} WINSTATIONCLIENTA, * PWINSTATIONCLIENTA;
```

```
#ifdef UNICODE
#define WINSTATIONCLIENT WINSTATIONCLIENTW
#define PWINSTATIONCLIENT PWINSTATIONCLIENTW
#else
#define WINSTATIONCLIENT WINSTATIONCLIENTA
#define PWINSTATIONCLIENT PWINSTATIONCLIENTA
#endif /* UNICODE */

/*
*  ICA specific protocol performance counters
*/

typedef struct  ICA COUNTERS {
    ULONG Reserved;
} ICA_COUNTERS, * PICA_COUNTERS;

/*
*  T.Share specific protocol performance counters
*/

typedef struct  TSHARE COUNTERS {
    ULONG Reserved;
} TSHARE COUNTERS, * PTSHARE COUNTERS;

/*
*  WinStation protocol performance counters
*/

typedef struct  PROTOCOLCOUNTERS {
    ULONG WdBytes;
    ULONG WdFrames;
    ULONG WaitForOutBuf;
    ULONG Frames;
    ULONG Bytes;
    ULONG CompressedBytes;
    ULONG CompressFlushes;
    ULONG Errors;
    ULONG Timeouts;
    ULONG AsyncFramingError;
    ULONG AsyncOverrunError;
    ULONG AsyncOverflowError;
    ULONG AsyncParityError;
    ULONG TdErrors;
    USHORT ProtocolType;
    USHORT Length;
    union {
        ICA COUNTERS    IcaCounters;
        TSHARE COUNTERS TShareCounters;
        ULONG           Reserved[100];
    } Specific;
} PROTOCOLCOUNTERS, * PPROTOCOLCOUNTERS;

/*
* ThinWire cache statistics
*/

typedef struct  THINWIRECACHE {
    ULONG CacheReads;
    ULONG CacheHits;
} THINWIRECACHE, * PTHINWIRECACHE;
#define MAX THINWIRECACHE   4

/*
*  ICA specific cache statistics
*/
```

```
typedef struct _ICA_CACHE {
    THINWIRECACHE ThinWireCache[ MAX_THINWIRECACHE ];
} ICA CACHE, * PICA CACHE;

/*
*  T.Share specific cache statistics
*/

typedef struct  TSHARE CACHE {
    ULONG Reserved;
} TSHARE CACHE, * PTSHARE CACHE;

/*
*  WinStation cache statistics
*/

typedef struct CACHE STATISTICS {
    USHORT ProtocolType;
    USHORT Length;
    union {
        ICA CACHE    IcaCacheStats;
        TSHARE CACHE TShareCacheStats;
        ULONG        Reserved[20];
    } Specific;
} CACHE_STATISTICS, * PCACHE_STATISTICS;

/*
*  WinStation protocol status
*/

typedef struct _PROTOCOLSTATUS {
    PROTOCOLCOUNTERS Output;
    PROTOCOLCOUNTERS Input;
    CACHE STATISTICS Cache;
    ULONG AsyncSignal;
    ULONG AsyncSignalMask;
} PROTOCOLSTATUS, * PPROTOCOLSTATUS;

#ifdef   cplusplus
typedef struct  PROTOCOLSTATUSEX : PROTOCOLSTATUS {
#else
typedef struct _PROTOCOLSTATUSEX {
    PROTOCOLSTATUS ;
#endif
    LARGE INTEGER  Counters[MAX COUNTER EXTENSIONS];
} PROTOCOLSTATUSEX, *PPROTOCOLSTATUSEX;

/*
*  WinStation query information
*/

typedef struct  WINSTATIONINFORMATIONW {
    WINSTATIONSTATECLASS ConnectState;
    WINSTATIONNAMEW WinStationName;
    ULONG LogonId;
    LARGE INTEGER ConnectTime;
    LARGE INTEGER DisconnectTime;
    LARGE INTEGER LastInputTime;
    LARGE INTEGER LogonTime;
    PROTOCOLSTATUS Status;
    WCHAR Domain[ DOMAIN_LENGTH + 1 ];
    WCHAR UserName[USERNAME_LENGTH + 1];
    LARGE INTEGER CurrentTime;
} WINSTATIONINFORMATIONW, * PWINSTATIONINFORMATIONW;
```

```
typedef struct _WINSTATIONINFORMATIONA {
    WINSTATIONSTATECLASS ConnectState;
    WINSTATIONNAMEA WinStationName;
    ULONG LogonId;
    LARGE INTEGER ConnectTime;
    LARGE INTEGER DisconnectTime;
    LARGE_INTEGER LastInputTime;
    LARGE_INTEGER LogonTime;
    PROTOCOLSTATUS Status;
    CHAR Domain[ DOMAIN LENGTH + 1 ];
    CHAR UserName[USERNAME LENGTH + 1];
    LARGE INTEGER CurrentTime;
} WINSTATIONINFORMATIONA, * PWINSTATIONINFORMATIONA;

#ifdef UNICODE
#define WINSTATIONINFORMATION WINSTATIONINFORMATIONW
#define PWINSTATIONINFORMATION PWINSTATIONINFORMATIONW
#else
#define WINSTATIONINFORMATION WINSTATIONINFORMATIONA
#define PWINSTATIONINFORMATION PWINSTATIONINFORMATIONA
#endif /* UNICODE */

/*
* Load balancing data types
*/

typedef enum  LOADFACTORTYPE {
    ErrorConstraint,
    PagedPoolConstraint,
    NonPagedPoolConstraint,
    AvailablePagesConstraint,
    SystemPtesConstraint,
    CPUConstraint
} LOADFACTORTYPE;

typedef struct  WINSTATIONLOADINDICATORDATA {
    ULONG RemainingSessionCapacity;
    LOADFACTORTYPE LoadFactor;
    ULONG TotalSessions;
    ULONG DisconnectedSessions;
    LARGE INTEGER IdleCPU;
    LARGE_INTEGER TotalCPU;
    ULONG RawSessionCapacity;
    ULONG reserved[9];
} WINSTATIONLOADINDICATORDATA, * PWINSTATIONLOADINDICATORDATA;

/*
*  WinStation shadow states
*/

typedef enum  SHADOWSTATECLASS {
    State NoShadow,
    State Shadowing,
    State_Shadowed,
} SHADOWSTATECLASS;

/*
*  Shadow query/set information
*/

typedef struct _WINSTATIONSHADOW {
    SHADOWSTATECLASS    ShadowState;
    SHADOWCLASS         ShadowClass;
    ULONG               SessionId;
```

```
    ULONG               ProtocolType;
} WINSTATIONSHADOW, * PWINSTATIONSHADOW;

typedef struct  WINSTATIONPRODIDW {
    WCHAR DigProductId[CLIENT PRODUCT ID LENGTH];
    WCHAR ClientDigProductId[CLIENT PRODUCT ID LENGTH ];
    WCHAR OuterMostDigProductId[CLIENT PRODUCT ID LENGTH ];
    ULONG curentSessionId;
    ULONG ClientSessionId;
    ULONG OuterMostSessionId;
}WINSTATIONPRODIDW, *PWINSTATIONPRODIDW;

typedef struct  WINSTATIONPRODIDA {
    CHAR DigProductId[CLIENT_PRODUCT_ID_LENGTH];
    CHAR ClientDigProductId[CLIENT_PRODUCT_ID_LENGTH ];
    CHAR OuterMostDigProductId[CLIENT PRODUCT ID LENGTH ];
    ULONG curentSessionId;
    ULONG ClientSessionId;
    ULONG OuterMostSessionId;
}WINSTATIONPRODIDA, *PWINSTATIONPRODIDA;

#ifdef UNICODE
#define WINSTATIONPRODID WINSTATIONPRODIDW
#define PWINSTATIONPRODID PWINSTATIONPRODIDW
#else
#define WINSTATIONPRODID WINSTATIONPRODIDA
#define PWINSTATIONPRODID PWINSTATIONPRODIDA
#endif /* UNICODE */

typedef struct {
    unsigned short sin family;
    union {
        struct {
            USHORT sin port;
            ULONG  in addr;
            UCHAR  sin zero[8];
        } ipv4;
        struct {
            USHORT sin6_port;
            ULONG  sin6 flowinfo;
            USHORT sin6 addr[8];
            ULONG  sin6 scope id;
        } ipv6;
    };
} WINSTATIONREMOTEADDRESS, *PWINSTATIONREMOTEADDRESS;

/*-------------------------------------------------*/
/*
*  Licensing Policy information struct
*/

#define LCPOLICYINFOTYPE V1 (1)
#define LCPOLICYINFOTYPE CURRENT LCPOLICYINFOTYPE V1

typedef struct {
    ULONG ulVersion;
    WCHAR * lpPolicyName;
    WCHAR * lpPolicyDescription;
} LCPOLICYINFO V1W, *LPLCPOLICYINFO V1W;

typedef struct {
    ULONG ulVersion;
    CHAR * lpPolicyName;
    CHAR * lpPolicyDescription;
} LCPOLICYINFO_V1A, *LPLCPOLICYINFO_V1A;
```

```
#ifdef UNICODE
#define LCPOLICYINFO_V1 LCPOLICYINFO_V1W
#define LPLCPOLICYINFO V1 LPLCPOLICYINFO V1W
#else
#define LCPOLICYINFO V1 LCPOLICYINFO V1A
#define LPLCPOLICYINFO V1 LPLCPOLICYINFO V1A
#endif
#define DEFAULT_POLICY_ID       1
#define PERSEAT POLICY ID       2
#define INTCONN POLICY ID       3
#define PERUSER POLICY ID       4
#define POLICY NOT CONFIGURED   5
#define MAXIMUM_POLICY_ID       6

/*-----------------------------------------------*/

typedef struct  BEEPINPUT {
    ULONG uType;
} BEEPINPUT, * PBEEPINPUT;

/*********************
*  NWLogon Structure
*********************/

#define IDTIMEOUT        32000
#define IDASYNC          32001
#define WSD LOGOFF      0x00000001
#define WSD SHUTDOWN    0x00000002
#define WSD REBOOT      0x00000004
#define WSD POWEROFF    0x00000008
#define WSD_FASTREBOOT  0x00000010

#define WTS CONSOLE CONNECT               0x1
#define WTS CONSOLE DISCONNECT            0x2
#define WTS REMOTE CONNECT                0x3
#define WTS REMOTE DISCONNECT             0x4
#define WTS_SESSION_LOGON                 0x5
#define WTS_SESSION_LOGOFF                0x6
#define WTS SESSION LOCK                  0x7
#define WTS SESSION UNLOCK                0x8
#define WTS SESSION REMOTE CONTROL        0x9

#define CREATE_MASK(__bit)   (1 << (__bit -1) )
#define WTS CONSOLE CONNECT MASK         CREATE MASK( WTS CONSOLE CONNECT )
#define WTS CONSOLE DISCONNECT MASK      CREATE MASK( WTS CONSOLE DISCONNECT )
#define WTS REMOTE CONNECT MASK          CREATE MASK( WTS REMOTE CONNECT )
#define WTS REMOTE DISCONNECT MASK       CREATE MASK( WTS REMOTE DISCONNECT )
#define WTS_SESSION_LOGON_MASK           CREATE_MASK( WTS_SESSION_LOGON )
#define WTS_SESSION_LOGOFF_MASK          CREATE_MASK( WTS_SESSION_LOGOFF )
#define WTS SESSION LOCK MASK            CREATE MASK( WTS SESSION LOCK )
#define WTS SESSION UNLOCK MASK          CREATE MASK( WTS SESSION UNLOCK )
#define WTS SESSION REMOTE CONTROL MASK  CREATE MASK( WTS SESSION REMOTE CONTROL )
#define WTS ALL NOTIFICATION MASK        0xFFFFFFFF

typedef struct _SESSIONDATAW {
    ULONG SessionId;
    WINSTATIONSTATECLASS State;
    ULONG Source;
    BOOLEAN bFullDesktop;
    GUID    SessionType;
    WINSTATIONNAMEW WinStationName;
    PROTOCOLSTATUS ProtocolStatus;
} SESSIONDATAW, * PSESSIONDATAW;
```

```
#define SESSIONDATA SESSIONDATAW
#define PSESSIONDATA PSESSIONDATAW

#ifndef  TS CONNECTIONTYPE GUID
#define  TS CONNECTIONTYPE GUID

//============================================================================
//
//Definitions of various terminal types.
//
//============================================================================

//GUID for Service Terminal
static const GUID TERMINAL_TYPE_SERVICE = /* 88f5767d-d13f-404d-a348-8b8e030294a9 */
{ 0x88f5767d, 0xd13f, 0x404d, { 0xa3, 0x48, 0x8b, 0x8e, 0x03, 0x02, 0x94, 0xa9 } };

//GUID for Regular desktop Remote Terminal
static const GUID TERMINAL TYPE REGULAR DESKTOP = /* 0f0a4bf8-8362-435d-938c-222a518a8b78
*/
{ 0x0f0a4bf8, 0x8362, 0x435d, { 0x93, 0x8c, 0x22, 0x2a, 0x51, 0x8a, 0x8b, 0x78 } };

//GUID for Remote Applications Terminal
static const GUID TERMINAL TYPE RDP REMOTEAPP = /* eddcc3ce-6e7e-4f4b-8439-3d9ad4c9440f
*/
{ 0xeddcc3ce, 0x6e7e, 0x4f4b, { 0x84, 0x39, 0x3d, 0x9a, 0xd4, 0xc9, 0x44, 0x0f } };

//GUID for MCE Terminal - This is same as MCE license type
static const GUID TERMINAL TYPE MCE = /* 8dc86f1d-9969-4379-91c1-06fe1dc60575 */
{ 0x8dc86f1d,0x9969,0x4379, { 0x91,0xc1,0x06,0xfe,0x1d,0xc6,0x05,0x75 } };

//GUID for the license pool that includes regular desktop, single app sessions,
//RAIL etc.
static const GUID LICENSE_TYPE_DEFAULT = /* 00000000-0000-0000-0000-000000000000  */
{ 0x00000000,0x0000,0x0000,{ 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 } };

static const GUID LICENSE TYPE BUILTIN = /* 45344fe7-00e6-4ac6-9f01-d01fd4ffadfb  */
{ 0x45344fe7,0x00e6,0x4ac6,{ 0x9f,0x01,0xd0,0x1f,0xd4,0xff,0xad,0xfb } };

//GUID for the app server license type.
static const GUID LICENSE TYPE APPSERVER = /* 36e64007-aef1-4085-89e3-66c26abade84  */
{ 0x36e64007,0xaef1,0x4085,{ 0x89,0xe3,0x66,0xc2,0x6a,0xba,0xde,0x84 } };

//GUID for the license pool that includes MCE sessions.

static const GUID LICENSE TYPE MCE = /* 8dc86f1d-9969-4379-91c1-06fe1dc60575 */
{ 0x8dc86f1d,0x9969,0x4379, { 0x91,0xc1,0x06,0xfe,0x1d,0xc6,0x05,0x75 } };

#endif  // _TS_CONNECTIONTYPE_GUID_
```

6.6 tsdef.h

For ease of implementation, the full header file is provided.

```
#ifdef __cplusplus
extern "C" {
#endif

typedef ULONG TNotificationId;

#define WTS_NOTIFY_NONE ( 0 )
```

```
#define WTS_NOTIFY_CREATE ( 0x1 )
#define WTS_NOTIFY_CONNECT ( 0x2 )
#define WTS_NOTIFY_DISCONNECT ( 0x4 )
#define WTS_NOTIFY_LOGON ( 0x8 )
#define WTS_NOTIFY_LOGOFF ( 0x10 )
#define WTS_NOTIFY_SHADOW_START ( 0x20 )
#define WTS_NOTIFY_SHADOW_STOP ( 0x40 )
#define WTS_NOTIFY_TERMINATE ( 0x80 )
#define WTS_NOTIFY_CONSOLE_CONNECT ( 0x100 )
#define WTS_NOTIFY_CONSOLE_DISCONNECT ( 0x200 )
#define WTS_NOTIFY_LOCK ( 0x400 )
#define WTS_NOTIFY_UNLOCK ( 0x800 )
#define WTS_NOTIFY_ALL ( 0xffffffff )

typedef enum _WINSTATIONUPDATECFGCLASS {
    WINSTACFG_LEGACY,
    WINSTACFG_SESSDIR
} WINSTATIONUPDATECFGCLASS;

typedef struct _SESSION_CHANGE
    {
    LONG SessionId;
    TNotificationId NotificationId;
    } SESSION_CHANGE;

typedef struct _SESSION_CHANGE *PSESSION_CHANGE;

#ifndef _TS_TIME_ZONE_INFORMATION_
#define _TS_TIME_ZONE_INFORMATION_
    typedef struct _TS_SYSTEMTIME {
        USHORT wYear;
        USHORT wMonth;
        USHORT wDayOfWeek;
        USHORT wDay;
        USHORT wHour;
        USHORT wMinute;
        USHORT wSecond;
        USHORT wMilliseconds;
    } TS_SYSTEMTIME;

    typedef struct _TS_TIME_ZONE_INFORMATION {
        LONG Bias;
        WCHAR StandardName[ 32 ];
        TS_SYSTEMTIME StandardDate;
        LONG StandardBias;
        WCHAR DaylightName[ 32 ];
        TS_SYSTEMTIME DaylightDate;
        LONG DaylightBias;
    } TS_TIME_ZONE_INFORMATION;

#endif //_TS_TIME_ZONE_INFORMATION_

    typedef enum _SESSION_FILTER {
        SF_SERVICES_SESSION_POPUP
    } SESSION_FILTER;

#ifdef __cplusplus
}
```

```
#endif
```

6.7 allproc.h

For ease of implementation, the full header file is provided.

```
#ifndef TS_ALLPROC_ALREADY_SET
#define TS_ALLPROC_ALREADY_SET

#ifdef  MIDL PASS
cpp quote( "#define TS PROCESS INFO MAGIC NT4  0x23495452" )
#else
#define TS_PROCESS_INFO_MAGIC_NT4  0x23495452
#endif

typedef ULONG PTR SIZE T;

typedef struct  TS PROCESS INFORMATION NT4 {
    ULONG MagicNumber;
    ULONG LogonId;
    PVOID ProcessSid;
    ULONG Pad;
} TS PROCESS INFORMATION NT4, * PTS PROCESS INFORMATION NT4;

// sizes of TS4.0 structures (size has changed in Windows 2000)
#ifdef  MIDL_PASS
cpp quote( "#define SIZEOF TS4 SYSTEM THREAD INFORMATION    64" )
cpp quote( "#define SIZEOF TS4 SYSTEM PROCESS INFORMATION   136" )
#else
#define SIZEOF TS4 SYSTEM THREAD INFORMATION 64
#define SIZEOF_TS4_SYSTEM_PROCESS_INFORMATION 136
#endif


#ifdef  MIDL PASS
cpp_quote( "#define GAP_LEVEL_BASIC 0" )
#else
#define GAP LEVEL BASIC 0
#endif

typedef struct  TS UNICODE STRING {
    USHORT Length;
    USHORT MaximumLength;
#ifdef MIDL_PASS
    [size is(MaximumLength),length is(Length)]PWSTR  Buffer;
#else
    PWSTR  Buffer;
#endif
} TS_UNICODE_STRING;


typedef struct  TS SYS PROCESS INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER SpareLi1;
    LARGE_INTEGER SpareLi2;
    LARGE INTEGER SpareLi3;
    LARGE INTEGER CreateTime;
    LARGE INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    TS_UNICODE_STRING ImageName;
```

```
    LONG BasePriority;                    // KPRIORITY in ntexapi.h
    DWORD UniqueProcessId;                // HANDLE in ntexapi.h
    DWORD InheritedFromUniqueProcessId;   // HANDLE in ntexapi.h
    ULONG HandleCount;
    ULONG SessionId;
    ULONG SpareUl3;
    SIZE T PeakVirtualSize;
    SIZE_T VirtualSize;
    ULONG PageFaultCount;
    ULONG PeakWorkingSetSize;
    ULONG WorkingSetSize;
    SIZE T QuotaPeakPagedPoolUsage;
    SIZE T QuotaPagedPoolUsage;
    SIZE_T QuotaPeakNonPagedPoolUsage;
    SIZE_T QuotaNonPagedPoolUsage;
    SIZE T PagefileUsage;
    SIZE T PeakPagefileUsage;
    SIZE T PrivatePageCount;
}
TS_SYS_PROCESS_INFORMATION, *PTS_SYS_PROCESS_INFORMATION;

typedef struct  TS ALL PROCESSES INFO {
    PTS SYS PROCESS INFORMATION pTsProcessInfo;
    DWORD                         SizeOfSid;
#ifdef MIDL_PASS
    [size_is(SizeOfSid)] PBYTE    pSid;
#else
    PBYTE                         pSid;
#endif
}
TS ALL PROCESSES INFO, *PTS ALL PROCESSES INFO;


//=============================================================================

// The following structures are defined for taking care of interface
// change in Whistler.

typedef struct _NT6_TS_UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
#ifdef MIDL PASS
    [size_is(MaximumLength / 2),length_is(Length / 2)]PWSTR  Buffer;
#else
    PWSTR  Buffer;
#endif
} NT6 TS UNICODE STRING;


typedef struct _TS_SYS_PROCESS_INFORMATION_NT6 {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE INTEGER SpareLi1;
    LARGE INTEGER SpareLi2;
    LARGE_INTEGER SpareLi3;
    LARGE_INTEGER CreateTime;
    LARGE INTEGER UserTime;
    LARGE INTEGER KernelTime;
    NT6 TS UNICODE STRING ImageName;
    LONG BasePriority;                    // KPRIORITY in ntexapi.h
    DWORD UniqueProcessId;                // HANDLE in ntexapi.h
    DWORD InheritedFromUniqueProcessId;   // HANDLE in ntexapi.h
    ULONG HandleCount;
    ULONG SessionId;
    ULONG SpareUl3;
```

```
    SIZE T PeakVirtualSize;
    SIZE_T VirtualSize;
    ULONG PageFaultCount;
    ULONG PeakWorkingSetSize;
    ULONG WorkingSetSize;
    SIZE T QuotaPeakPagedPoolUsage;
    SIZE T QuotaPagedPoolUsage;
    SIZE_T QuotaPeakNonPagedPoolUsage;
    SIZE_T QuotaNonPagedPoolUsage;
    SIZE T PagefileUsage;
    SIZE T PeakPagefileUsage;
    SIZE T PrivatePageCount;
}
TS_SYS_PROCESS_INFORMATION_NT6, *PTS_SYS_PROCESS_INFORMATION_NT6;

typedef struct  TS ALL PROCESSES INFO NT6 {
    PTS SYS PROCESS INFORMATION NT6 pTsProcessInfo;
    DWORD                           SizeOfSid;
#ifdef MIDL PASS
    [size_is(SizeOfSid)] PBYTE      pSid;
#else
    PBYTE                           pSid;
#endif
}
TS_ALL_PROCESSES_INFO_NT6, *PTS_ALL_PROCESSES_INFO_NT6;

//=============================================================================================
======

//
// TermSrv Counter Header
//
typedef struct _TS_COUNTER_HEADER {
    DWORD dwCounterID;     // identifies counter
    BOOLEAN bResult;       // result of operation performed on counter
} TS COUNTER HEADER, *PTS COUNTER HEADER;

typedef struct _TS_COUNTER {
    TS_COUNTER_HEADER counterHead;
    DWORD             dwValue;      // returned value
    LARGE INTEGER     startTime;    // start time for counter
} TS COUNTER, *PTS COUNTER;

#endif  //  TS_ALLPROC_ALREADY_SET

//NBD   end
```

7 Appendix B: Windows Behavior

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

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

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

<1> Section 1.8: Windows only uses the values specified in [MS-ERREF].

<2> Section 2.2.1.12: On Windows Vista and Windows Server 2008, WinStationConnectState is not defined; instead, the following two enum values are used.

Value	Meaning
WinStationReconnectedFromId	In case of reconnected sessions, this will return the session ID of the temp session from which it was reconnected, or -1 if there is no temp session.
WinStationEffectsPolicy	SKU-Differentiation policy for the DWM-over-TS.

<3> Section 2.2.1.12: Not used in Windows Vista, Windows Server 2008 or later system releases.

<4> Section 2.2.1.13: This state is not supported in Windows Vista and Windows Server 2008.

<5> Section 2.2.2.3: This field is only available in Windows Vista and Windows Server 2008 or later system releases.

<6> Section 2.2.2.11.1: Supported on Windows XP only:

Current number of disconnected desktop sessions: **dwCounterId** should be TERMSRV_CURRENT_DISC_DESKTOP_SESSIONS. Value will indicate the current number of disconnected desktop (normal) sessions.

Current number of disconnected subdesktop sessions: **dwCounterId** should be TERMSRV_CURRENT_DISC_SUBDESKTOP_SESSIONS. Value will indicate the current number of disconnected subdesktop sessions.

Current number of active desktop sessions: **dwCounterId** should be TERMSRV_CURRENT_ACTIVE_DESKTOP_SESSIONS. Value will indicate the current number of active desktop (normal) sessions.

Current number of disconnected subdesktop sessions: **dwCounterId** should be TERMSRV_CURRENT_ACTIVE_SUBDESKTOP_SESSIONS. Value will indicate the current number of active subdesktop sessions.

Added in Windows Server 2003:

Current number of pending sessions: **dwCounterId** should be TERMSRV_PENDING_SESSIONS. Value will indicate the current number of pending connections to the server.

Total number of successful logons: **dwCounterId** should be TERMSRV_SUCC_TOTAL_LOGONS. Value will indicate the total number of successful logons on the server, both locally and remotely.

Total number of successful local logons: **dwCounterId** should be TERMSRV_SUCC_LOCAL_LOGONS. Value will indicate the total number of successful local logons on the server.

Total number of successful remote logons: **dwCounterId** should be TERMSRV_SUCC_REMOTE_LOGONS. Value will indicate the total number of successful remote logons on the server.

Total number of successful session 0 logons: **dwCounterId** should be TERMSRV_SUCC_SESSION0_LOGONS. Value will indicate the total number of successful connects on the server to session 0.

Added in Windows Vista and Windows Server 2008:

Number of terminating sessions: **dwCounterId** should be TERMSRV_CURRENT_TERMINATING_SESSIONS. Value will indicate the current number of terminating sessions on the server.

Number of logged on sessions: **dwCounterId** should be TERMSRV_CURRENT_LOGGEDON_SESSIONS. Value will indicate the current number of logged-on sessions on the server.

<7> Section 2.2.2.12.1: This has been deprecated in Windows Vista and Windows Server 2008.

<8> Section 2.2.2.12.1: Not used in Windows Vista and Windows Server 2008.

<9> Section 2.2.2.12.1: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<10> Section 2.2.2.12.1: Added in Windows XP and supported from Windows XP on.

<11> Section 2.2.2.12.1: Added in Windows XP and supported from Windows XP on. The supported values 1, 2, 4, 8, 16 are translated as the following number of colors supported: 16 (4 bpp), 256 (8 bpp), 65,536 (16 bpp), 16 million (24 bpp), 32,768 (15 bpp), respectively. On Windows Vista and Windows Server 2008 the value 32 was added to indicate 32-bit color.

<12> Section 2.2.2.12.1: Added in Windows XP and supported from Windows XP on.

<13> Section 2.2.2.12.1: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<14> Section 2.2.2.12.1: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<15> Section 2.2.2.12.2: This has been deprecated in Windows Vistaand Windows Server 2008.

<16> Section 2.2.2.12.2: Not used in Windows Vista and Windows Server 2008.

<17> Section 2.2.2.12.2: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<18> Section 2.2.2.12.2: Added in Windows XP and supported from Windows XP on.

<19> Section 2.2.2.12.2: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<20> Section 2.2.2.12.2: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<21> Section 2.2.2.13.1: Added in Windows XP and supported from Windows XP on.

<22> Section 2.2.2.13.1: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<23> Section 2.2.2.13.1: Added in Windows Vista and supported from Windows Vista and Windows Server 2008 on.

<24> Section 2.2.2.13.1: IPv6 is supported on Windows Vista and Windows Server 2008 only.

<25> Section 2.2.2.13.1: On Windows Vista and Windows Server 2008 the value 32 was added to indicate 32-bit color.

<26> Section 2.2.2.13.1: PROTOCOL_CONSOLE was added on Windows XP and is supported from Windows XP on.

<27> Section 2.2.2.13.1: Added in Windows XP and supported from Windows XP on.

<28> Section 2.2.2.13.1: Added in Windows XP and supported from Windows XP on.

<29> Section 2.2.2.13.1: Added in Windows XP and supported from Windows XP on. If the Terminal Server client is either RDP 5.0, 5.1 or 6.0 and is running on Windows XP/Windows Server 2003, this field contains client machine's product ID from registry HKLM\Software\microsoft\windows\currentversion\productID. If RDP 6.0 Terminal Server client is running on a Windows Vista machine, it contains a unique instance ID of LSM process which is InstanceID in the TS registry

<30> Section 2.2.2.13.1: Added in Windows XP and supported from Windows XP on.

<31> Section 2.2.2.13.1: Windows Vista, Windows Server 2008, and later only.

<32> Section 2.2.2.13.1: Windows Vista, Windows Server 2008, and later only.

<33> Section 2.2.2.13.1: (Windows 2000 RDP client)

<34> Section 2.2.2.13.1: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<35> Section 2.2.2.13.2: Added in Windows XP and supported from Windows XP on.

<36> Section 2.2.2.13.2: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<37> Section 2.2.2.13.2: Added in Windows Vista and supported from Windows Vista and Windows Server 2008on.

<38> Section 2.2.2.13.2: IPv6 is supported on Windows Vista and Windows Server 2008 only.

<39> Section 2.2.2.13.2: On Windows Vista and Windows Server 2008 the value 32 was added to indicate 32-bit color.

<40> Section 2.2.2.13.2: PROTOCOL_CONSOLE was added on Windows XP and is supported from Windows XP on.

<41> Section 2.2.2.13.2: Added in Windows XP and supported from Windows XP on.

<42> Section 2.2.2.13.2: Added in Windows XP and supported from Windows XP on.

<43> Section 2.2.2.13.2: Added in Windows XP and supported from Windows XP on. If the Terminal Serverclient is either RDP 5.0, 5.1 or 6.0 and is running on Windows XP/Windows Server 2003, this field contains the client machine's product ID from registry HKLM\Software\microsoft\windows\currentversion\productID. If the RDP 6.0 Terminal Serverclient is running on Windows Vista machine, it contains a unique instance ID of the LSM process, which is InstanceID in the Terminal Server registry.

<44> Section 2.2.2.13.2: Added in Windows XP and supported from Windows XP on.

<45> Section 2.2.2.13.2: Windows Vista, Windows Server 2008 and beyond only.

<46> Section 2.2.2.13.2: Windows Vista, Windows Server 2008 and beyond only.

<47> Section 2.2.2.13.2: Windows 2000 RDP client.

<48> Section 2.2.2.13.2: Added in Windows Server 2003 and supported from Windows Server 2003 on.

<49> Section 2.2.2.14.3.2.1: Not used by Microsoft Terminal Services.

<50> Section 2.2.2.14.3.3.1: Not used by Microsoft Terminal Services.

<51> Section 2.2.2.30: PROTOCOL_CONSOLE is supported on Windows XP, Windows Storage Server 2003, Windows Vista, and Windows Server 2008.

<52> Section 2.2.2.32: IPv6 format addresses are only supported on Windows Vista and Windows Server 2008.

<53> Section 3.6.4.1: Opnums reserved for local use apply to Windows as follows.

Opnum	Description
20 - 25	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
34	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
38	Only used locally by Windows, never remotely.
39 - 41	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
50 - 52	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
63 - 64	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
67 - 68	Server returns ERROR_NOT_IMPLEMENTED. It is never used.

<54> Section 3.6.4.1.3: This call is NOT supported in Windows Vista and Windows Server 2008.

<55> Section 3.6.4.1.5: This call is NOT supported in Windows Vista and Windows Server 2008.

<56> Section 3.6.4.1.7: This call is NOT supported on Windows Vista and Windows Server 2008.

<57> Section 3.6.4.1.7: Ignored by the Terminal Server RDP drivers in Windows NT Server 4.0, Windows 2000, Windows XP, and Windows Server 2003.

<58> Section 3.6.4.1.7: Ignored by the Terminal Server RDP drivers in Windows NT Server 4.0, Windows 2000, Windows XP, and Windows Server 2003.

<59> Section 3.6.4.1.7: Supported on Windows XP and Windows Server 2003 only.

<60> Section 3.6.4.1.11: This parameter has no effect on Windows Vista and the Windows Server 2008.

<61> Section 3.6.4.1.12: This call is NOT supported on Windows Vista and Windows Server 2008.

<62> Section 3.6.4.1.14: This parameter has no effect on Windows Vista and the Windows Server 2008.

<63> Section 3.6.4.1.15: In the case of resetting a listener, all session originated from the listener will be disconnected in Windows Vista and Windows Server 2008.

<64> Section 3.6.4.1.18: This call is NOT supported in Windows Vista and Windows Server 2008.

<65> Section 3.6.4.1.19: It is not necessary to call this method on Windows Vista or the Windows Server 2008 because the setting up of the session for shadowing is included in the functionality of **RpcWinStationShadowTarget**.

<66> Section 3.6.4.1.19: The implementation of this method on Windows Vista and the Windows Server 2008 always returns TRUE.

<67> Section 3.6.4.1.21: This call is NOT supported on Windows Vista and Windows Server 2008.

<68> Section 3.6.4.1.22: This call is NOT supported on Windows Vista and Windows Server 2008.

<69> Section 3.6.4.1.23: This call is NOT supported on Windows Vista and Windows Server 2008.

<70> Section 3.6.4.1.24: This call is NOT supported on Windows Vista, Windows Server 2008, Windows Server 2003, and Windows XP.

<71> Section 3.6.4.1.24: On Windows 2000, the method ignores any value for *LogonId* other than -2 and will not allow the debugger to be broken into in kernel mode or in a particular session's csrss.exe process.

<72> Section 3.6.4.1.25: In the case of resetting a listener, all sessions originated from the listener will be disconnected in Windows Vista and Windows Server 2008.

<73> Section 3.6.4.1.26: In the case of resetting a listener, all sessions originated from the listener will be disconnected in Windows Vista and Windows Server 2008.

<74> Section 3.6.4.1.27: In the case of resetting a listener, all sessions originated from the listener will be disconnected in Windows Vista and Windows Server 2008.

<75> Section 3.6.4.1.27: Added in Windows Server 2003.

<76> Section 3.6.4.1.28: This call is not supported in Windows Vista and Windows Server 2008.

<77> Section 3.6.4.1.29: The implementation of this method is dependent on the Terminal Server protocol-specific extension DLL implementing the method WsxWinStationLogonAnnoyance [WININTERNALS]. That method is not implemented in the built-in RDP implementation from Windows 2000on. Winlogon itself only calls this method in Terminal Server on Windows NT Server 4.0.

<78> Section 3.6.4.1.29: This call is not supported in Windows Vista and Windows Server 2008.

<79> Section 3.6.4.1.29: On Windows Vista and the Windows Server 2008 this method always returns TRUE.

<80> Section 3.6.4.1.30: The **RpcWinStationEnumerateProcesses** method returns the process information for a Windows NT Server 4.0 Terminal Server and is supported only for backwards compatibility with that platform.

<81> Section 3.6.4.1.32: Supported on Windows NT 4.0 Terminal Server 4.0 only.

<82> Section 3.6.4.1.36: This call is NOT supported on Windows Vista and Windows Server 2008.

<83> Section 3.6.4.1.37: This call is NOT supported on Windows Vista and Windows Server 2008.

<84> Section 3.6.4.1.37: Specific to Windows NT.

<85> Section 3.6.4.1.38: This call is NOT supported on Windows Vista and Windows Server 2008.

<86> Section 3.6.4.1.39: This call is NOT supported in Windows Vista and Windows Server 2008.

<87> Section 3.6.4.1.40: This call is NOT supported in Windows Vista and Windows Server 2008 and Windows Server 2008.

<88> Section 3.6.4.1.41: This call is NOT supported on Windows Vista.

<89> Section 3.6.4.1.42: This call is NOT supported on Windows Vista and Windows Server 2008.

<90> Section 3.6.4.1.43: This call is NOT supported on Windows Vista and Windows Server 2008.

The method definition is for Windows Server 2003. Windows XP does not have the *dwMask* parameter, and the session identified by SessionId will receive all notifications.

<91> Section 3.6.4.1.44: This call is NOT supported on Windows Vista and Windows Server 2008.

<92> Section 3.6.4.1.45: This call is NOT supported on Windows Vista and Windows Server 2008.

On Windows XP, a class of WINSTACFG_SESSDIR will do nothing and always returns success.

<93> Section 3.6.4.1.46: This call is NOT supported on Windows Vista and Windows Server 2008.

<94> Section 3.6.4.1.48: This call is NOT supported on Windows Vista and Windows Server 2008.

<95> Section 3.6.4.1.49: This call is NOT supported on Windows Vista and Windows Server 2008.

<96> Section 3.6.4.1.50: This call is NOT supported on Windows Vista and Windows Server 2008.

<97> Section 3.6.4.1.51: This call is NOT supported on Windows Vista and Windows Server 2008.

<98> Section 3.6.4.1.52: This call is NOT supported on Windows Vista and Windows Server 2008.

<99> Section 3.6.4.1.54: Supported on Windows Server 2003 only.

<100> Section 3.6.4.1.55: This call is NOT supported on Windows Vista and Windows Server 2008.

Supported on Windows Server 2003 only.

<101> Section 3.6.4.1.56: This call is NOT supported on Windows Vista and Windows Server 2008.

<102> Section 3.6.4.1.57: Supported on Windows Server 2003 only.

<103> Section 3.6.4.1.58: Supported on Windows Server 2003 only, and then only on advanced servers running in application server mode.

<104> Section 3.8.4.1: Opnums reserved for local use apply to Windows as follows.

Opnum	Description
2	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
3	Server returns ERROR_NOT_IMPLEMENTED. It is never used.
8	Server returns ERROR_NOT_IMPLEMENTED. It is never used.

8 Index

M

N

O

P

Q

R

S

T

U

V

W