[MS-W32T]:
W32Time Remote Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	0.1.1	Editorial	Revised and edited the technical content.
10/23/2007	0.1.2	Editorial	Revised and edited the technical content.
11/30/2007	0.2	Minor	Added statement about default behavior.
01/25/2008	0.3	Minor	Updated the technical content.

Table of Contents

1 Introduction

The W32Time Remote Protocol is a Microsoft proprietary **remote procedure call (RPC)** interface for controlling and monitoring a **time service** on a machine. This RPC interface supports time services that synchronize time using the Network Time Protocol (NTP) Version 3, as specified in [RFC1305], as well as platform-specific hardware **time sources**.

1.1 Glossary

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

Authentication Level
Endpoint
Error Code
Message Identifier
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Opnum
Primary Domain Controller (PDC)
Reliable Time Source
Remote Procedure Call (RPC)
RPC Protocol Sequence
Server Message Block (SMB)
Time Peer
Time Provider
Time Service
Time Source
Universal Unique Identifier (UUID) or Globally Unique Identifier (GUID)
UTC (Coordinated Universal Time)
Well-Known Endpoint
Windows Time Service (W32Time)

The following terms are specific to this document:

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

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

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

[MS-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.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", January 2007.

[RFC1305] Mills, D. L., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992, http://www.ietf.org/rfc/rfc1305.txt

[RFC1769] Mills, D., "Simple Network Time Protocol (SNTP)", RFC 1769, March 1995, http://www.ietf.org/rfc/rfc1769.txt

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

1.2.2 Informative References

[MS-SNTP] Microsoft Corporation, "Network Time Protocol (NTP) Authentication Extensions", March 2007.

[NTP-TR9733] Mills, D., "Clock Discipline Algorithms for the Network Time Protocol Version 4", March 1997, http://www.ee.udel.edu/~mills/database/reports/allan/secureb.pdf

[NTP-TR9733i] Mills, D., "Clock Discipline Algorithms for the Network Time Protocol Version 4", March 1997, http://www.ee.udel.edu/~mills/database/reports/allan/securea.pdf

[WTSREF] Microsoft Corporation, "Windows Time Service Technical Reference", March 2003, http://technet2.microsoft.com/WindowsServer/en/Library/a0fcd250-e5f7-41b3-b0e8-240f8236e2101033.mspx

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

1.3 Protocol Overview (Synopsis)

The W32Time Remote Protocol provides an RPC interface used for controlling and monitoring a time service. The client end of the W32Time Remote Protocol is an application that issues method calls on the RPC interface. The server end of the W32Time Remote Protocol is a time service that implements support for this RPC interface.<1>

1.4 Relationship to Other Protocols

The W32Time Remote Protocol depends on RPC for its transport. The protocol can be used to control time services, including time services that implement the Network Time Protocol (NTP) Version 3 (as specified in [RFC1305]) and the Simple Network Time Protocol (SNTP) (as specified in [RFC1769]).

The W32Time Remote Protocol is commonly used to control and monitor the **Windows Time service (W32Time)**, a time service that implements support for NTP on computers running Windows. However, the Windows implementation of NTP is not dependent on the W32Time Remote Protocol. For more information on the Windows implementation of NTP, see [MS-SNTP].

1.5 Prerequisites/Preconditions

This protocol is an RPC interface, and therefore has the prerequisites common to RPC interfaces, as specified in [MS-RPCE].

It is assumed that a W32Time Remote Protocol client has obtained the name of a server that supports both a time service and the W32Time Remote Protocol before this protocol is invoked.

1.6 Applicability Statement

This protocol is applicable wherever there is a need to control or monitor time services. The W32Time Remote Protocol does not participate in time synchronization.

1.7 Versioning and Capability Negotiation

- **Supported Transports:** This protocol uses RPC over **Server Message Block (SMB)**, as specified in [MS-SMB], as its only supported transport. For supported transports, see section 2.1.
- **Protocol Version:** This protocol's RPC interface has a single version number of 4.1. This protocol may be extended without altering the version number by adding RPC methods to the interface with **opnums** lying numerically beyond those defined in this specification. A client determines whether such methods are supported by attempting to invoke the method; if the method is not supported, the RPC server MUST return an "opnum out of range" error, as specified in [C706] and [MS-ERREF]. For the RPC interface, see [MS-RPCE].<2>
- **Security and Authentication Methods:** For security considerations, see sections 3.1.3 and 3.2.3.

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

Parameter	Value	Reference
RPC interface **UUID**	8fb6d884-2388-11d0-8c35-00c04fda2795	Section 2.1
Pipe name	\\PIPE\W32TIME	Section 2.1
Pipe name	\\PIPE\W32TIME_ALT	Section 2.1

2 Messages

The following sections specify how W32Time Remote Protocol messages are transported and W32Time Remote Protocol message syntax.

2.1 Transport

This protocol MUST use the following **RPC protocol sequence**: RPC over SMB, as specified in [MS-RPCE] section 2.1.1.2.

This protocol MUST use the following **well-known endpoints**. These **endpoints** are pipe names for RPC over SMB, as specified in [MS-RPCE] section 2.1.1.2.

- \\PIPE\W32TIME: This endpoint MUST be used for the unauthenticated RPC interface.
- \\PIPE\W32TIME_ALT: This endpoint MUST be used for the authenticated RPC interface. The authenticated RPC interface allows RPC to negotiate the use of authentication and the **authentication level** on behalf of the client and server, as specified in [MS-RPCE] sections 2.2.2.11 and 5.1.1.<3>

This protocol MUST use the UUID as specified in section 1.9. The RPC version number is 4.1.

2.2 Common Data Types

In addition to the RPC base types and definitions specified in [C706] and [MS-RPCE], the following data types are defined in the **Microsoft Interface Definition Language (MIDL)** specification for this RPC interface.

The types used in the IDL fragments in the following data type definitions are specified in [MS-RPCE] section 2.2.

Only the values defined in the following data type definitions SHOULD be used. All other values are reserved for future use, and SHOULD NOT be used.

2.2.1 W32TIME_PROVIDER_INFO

The **W32TIME_PROVIDER_INFO** structure defines information about a selected **time provider** (either an NTP time provider or a hardware time provider).

```
typedef struct {
  unsigned __int32 ulProviderType;
  [switch_is(ulProviderType)] W32TIME_PROVIDER_DATA ProviderData;
} W32TIME_PROVIDER_INFO,
 *PW32TIME_PROVIDER_INFO;
```

ulProviderType: The type of time provider, which MUST be one of the following values.

All other values are reserved for future use and servers SHOULD NOT send them. Clients receiving an unknown **ulProviderType** value MAY ignore it.<4>

Value	Meaning
0	NTP time provider
1	Hardware time provider

ProviderData: A **W32TIME_PROVIDER_DATA** union that contains information about the time provider.

2.2.2 W32TIME_PROVIDER_DATA

The **W32TIME_PROVIDER_DATA** union selects either an NTP time provider or a hardware time provider.

```
typedef
[switch_type(unsigned __int32)]
union {
  [case(0)]
    W32TIME_NTP_PROVIDER_DATA* pNtpProviderData;
  [case(1)]
    W32TIME_HARDWARE_PROVIDER_DATA* pHardwareProviderData;
} W32TIME_PROVIDER_DATA;
```

pNtpProviderData: A **W32TIME_NTP_PROVIDER_DATA** structure that contains information about an NTP time provider.

pHardwareProviderData: A **W32TIME_HARDWARE_PROVIDER_DATA** structure that contains information about a hardware time provider. Examples of hardware time providers include cesium and atomic clocks.

2.2.3 W32TIME_HARDWARE_PROVIDER_DATA

The **W32TIME_HARDWARE_PROVIDER_DATA** structure contains operational information about a hardware time provider, such as a cesium or atomic clock.

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulError;
  unsigned __int32 ulErrorMsgId;
  [string, unique] wchar_t* wszReferenceIdentifier;
} W32TIME_HARDWARE_PROVIDER_DATA,
 *PW32TIME_HARDWARE_PROVIDER_DATA;
```

ulSize: The size, in bytes, of this structure.

ulError: An **error code** that indicates success or failure. A value of 0 MUST be used to indicate that the hardware time provider is functioning correctly. A nonzero value MUST be used to indicate the hardware time provider's failure. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for the purposes of this protocol.<5>

ulErrorMsgId: An integer that maps to a **message identifier** for a message that SHOULD describe the failure indicated in the **ulError** field. The values in this field are implementation-specific. If an implementation receives a value it does not understand, the implementation MUST ignore the value.<6>

wszReferenceIdentifier: The Reference Clock Identifier that identifies the time source for this time service, as specified in [RFC1305] Appendix A, "NTP Data Format".

2.2.4 W32TIME_NTP_PROVIDER_DATA

The **W32TIME_NTP_PROVIDER_DATA** structure defines the state of an NTP time provider.

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulError;
  unsigned __int32 ulErrorMsgId;
  unsigned __int32 cPeerInfo;
  [size_is(cPeerInfo)] W32TIME_NTP_PEER_INFO pPeerInfo[];
} W32TIME_NTP_PROVIDER_DATA,
 *PW32TIME_NTP_PROVIDER_DATA;
```

ulSize: The size, in bytes, of this structure.

ulError: An error code that indicates success or failure. A value of 0 MUST indicate that the NTP time provider is functioning correctly. A nonzero value MUST indicate that the NTP time provider has encountered a failure. Because the values transmitted in this field are implementation-specific, all nonzero values MUST be treated as equivalent for protocol purposes.<7>

ulErrorMsgId: An integer that maps to a message identifier for a human-readable message that SHOULD describe the failure indicated in the **ulError** field. The values in this field are implementation-specific. If an implementation receives a value it does not understand, the implementation MUST ignore the value.<8>

cPeerInfo: The number of active **time peers** that synchronize with this NTP time provider. This value also indicates the number of structures in **pPeerInfo**.

pPeerInfo: An array of **W32TIME_NTP_PEER_INFO** structures representing the time peers with which this time provider is currently synchronizing.

2.2.5 W32TIME_NTP_PEER_INFO

The **W32TIME_NTP_PEER_INFO** structure defines the current state of a time peer for an NTP time provider.

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulResolveAttempts;
  unsigned __int64 u64TimeRemaining;
  unsigned __int64 u64LastSuccessfulSync;
  unsigned __int32 ulLastSyncError;
  unsigned __int32 ulLastSyncErrorMsgId;
  unsigned __int32 ulValidDataCounter;
  unsigned __int32 ulAuthTypeMsgId;
```

```
  [string, unique] wchar_t* wszUniqueName;
  unsigned char ulMode;
  unsigned char ulStratum;
  unsigned char ulReachability;
  unsigned char ulPeerPollInterval;
  unsigned char ulHostPollInterval;
} W32TIME_NTP_PEER_INFO,
 *PW32TIME_NTP_PEER_INFO;
```

ulSize: The size, in bytes, of this structure, including the length of the string that **wszUniqueName** points to.<9>

ulResolveAttempts: The number of consecutive times the NTP time provider attempted to resolve this time peer but was not successful. A value of 0 MUST indicate that the last attempt to resolve the peer was successful. Resolution MUST include time source discovery and DNS name resolution, as specified in [RFC1305]. The value SHOULD NOT<10> wrap around to 0.

u64TimeRemaining: The amount of time remaining until the time provider repolls this time peer, in 100 nanosecond units.

u64LastSuccessfulSync: The time that has passed since this time peer was last successfully synchronized, expressed as the number of 100-nanosecond intervals since midnight January 1, 1601, in **Coordinated Universal Time (UTC)**.

ulLastSyncError: An error code that indicates success or failure of the last attempt to synchronize time with this time peer. A value of 0 MUST indicate success. A nonzero value MUST indicate failure. Because the values transmitted in this field are implementation-specific, all nonzero values MUST be treated as equivalent for the purposes of this protocol.<11>

ulLastSyncErrorMsgId: An integer that maps to a message identifier for a message that SHOULD describe the failure indicated in the **ulLastSyncError** field. The values in this field are implementation-specific. If an implementation receives a value it does not understand, the implementation MUST ignore the value.<12>

ulValidDataCounter: The number of valid time samples from this time peer that the NTP time provider currently has stored in its database.

ulAuthTypeMsgId: An integer that maps to a message identifier for a message that describes the authentication mechanism that this time peer uses for secure NTP communication. For more information on NTP authentication, see [WTSREF] and [MS-SNTP]. The values in this field are implementation-specific. If an implementation receives a value it does not understand, the implementation MUST ignore the value.<13>

wszUniqueName: A name that identifies this time peer. This name is for informational or display purposes only and is implementation-specific. This field MAY<14> be ignored for protocol purposes.

ulMode: This time peer's current NTP association mode, as specified in [RFC1305] section 3.3, "Modes of Operation".

ulStratum: This time peer's stratum level, which indicates the distance between this time peer and a reference source. This value is compared with other peers' stratum levels to ensure that a machine closer to a reference source is not synchronized to a machine that is farther away, as specified in [RFC1305] section 2.2, "Network Configurations".

ulReachability: An 8-bit shift register that contains this time peer's reachability, as specified in [RFC1305] section 3.2.3, "Peer Variables".

ulPeerPollInterval: This time peer's poll interval, expressed as specified in [RFC1305], using units of seconds given as exponents to a power of two.

ulHostPollInterval: The interval at which the NTP service provider is polling this time peer, expressed as specified in [RFC1305], using units of seconds given as exponents to a power of two.

2.2.6 Type of Time Service Configuration Setting

The type of configuration setting represents how each setting of the time provider or time service is configured. The following table lists the configuration-setting types that are available.

Value	Meaning
W32TIME_CONFIGURATION_SETTING_UNDEFINED 0x00000000	The configuration setting is not defined.
W32TIME_CONFIGURATION_SETTING_DEFAULT 0x00000001	The configuration setting is using the default value.
W32TIME_CONFIGURATION_SETTING_LOCAL 0x00000002	The configuration setting is defined locally.
W32TIME_CONFIGURATION_SETTING_POLICY 0x00000003	The configuration setting is defined remotely, such as through Group Policy.
W32TIME_CONFIGURATION_SETTING_RESERVED 0x00000004	The configuration setting is reserved.

All other values are reserved for future use.

2.2.7 State of Time Service

The state of the time service represents the current state of the time service in the clock discipline algorithm. For more information on clock discipline algorithms, see [NTP-TR9733i] and [NTP-TR9733] section 4.3.

The clock discipline algorithm is optional and implementation-specific. The following table lists the values that MAY<15> be supported.

Value	Meaning
UNSET 0x00000000	The clock is not synchronized.
HOLD 0x00000001	The clock is adjusting the time difference only, not the clock rate.
SYNC 0x00000002	The clock is synchronized. Both the time difference and the clock rate are adjusted.

Value	Meaning
SPIKE 0x00000003	A spike was detected, but the clock discipline algorithm cannot determine if the clock is no longer synchronized or if the spike was just a network jitter.

All other values are reserved for future use.

2.2.8 W32TIME_CONFIGURATION_PROVIDER

The **W32TIME_CONFIGURATION_PROVIDER** structure defines the configuration data of an NTP time provider.

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulInputProvider;
  unsigned __int32 ulEnabled;
  [string, unique] wchar_t* wszDllName;
  [string, unique] wchar_t* wszProviderName;
  unsigned __int32 ulDllNameFlag;
  unsigned __int32 ulProviderNameFlag;
  unsigned __int32 ulInputProviderFlag;
  unsigned __int32 ulEnabledFlag;
  PW32TIME_PROVIDER_CONFIG pProviderConfig;
} W32TIME_CONFIGURATION_PROVIDER,
 *PW32TIME_CONFIGURATION_PROVIDER;
```

ulSize: The size, in bytes, of this structure.

ulInputProvider: An integer that indicates whether or not the provider is an input provider. A provider responding to NTP client requests becomes an NTP server, as specified in [RFC1305].

Value	Meaning
0	MUST indicate that the provider is not an input provider.
1	MUST indicate that the provider is an input provider.

All other values are reserved for future use.

ulEnabled: An integer that indicates whether or not the provider is enabled.

Value	Meaning
0	MUST indicate that the provider is disabled.
1	MUST indicate that the provider is enabled.

All other values are reserved for future use.

wszDllName: A null-terminated string indicating the location of the DLL in which the provider is implemented. The location is represented as the full path of the DLL's file name in the file system.

wszProviderName: A null-terminated string indicating the name of the provider.

ulDllNameFlag: An integer indicating the type of the configuration setting for **wszDllName**, as specified in section 2.2.6.

ulProviderNameFlag: An integer indicating the type of the configuration setting for **wszProviderName**, as specified in section 2.2.6.

ulInputProviderFlag: An integer indicating the type of the configuration setting for **ulInputProvider**, as specified in section 2.2.6.

ulEnabledFlag: An integer indicating the type of the configuration setting for **ulEnabled**, as specified in section 2.2.6.

pProviderConfig: A pointer to the **W32TIME_PROVIDER_CONFIG** structure.

2.2.9 W32TIME_PROVIDER_CONFIG

The **W32TIME_PROVIDER_CONFIG** structure defines configuration data for a selected time provider.

```
typedef struct {
  unsigned   int32 ulSize;
  unsigned   int32 ulProviderType;
  [switch is(ulProviderType)] PW32TIME PROVIDER CONFIG DATA pProviderConfigData;
} W32TIME_PROVIDER_CONFIG,
 *PW32TIME_PROVIDER_CONFIG;
```

ulSize: The size, in bytes, of this structure.

ulProviderType: The type of time provider, which MUST be one of the following values.

Value	Meaning
W32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA 0x00000000	NtpClient NTP time provider
W32TIME_NTPSERVER_PROVIDER_CONFIG_DATA 0x00000001	NtpServer NTP time provider

All other values are reserved for future use.

pProviderConfigData: A **W32TIME_PROVIDER_CONFIG_DATA** union that contains configuration data about the time provider.

2.2.10 W32TIME_PROVIDER_CONFIG_DATA

The **W32TIME_PROVIDER_CONFIG_DATA** union selects either an NtpClient or an NtpServer time provider.

```
typedef
[switch_type(unsigned __int32)]
union {
  [case(0)]
    PW32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA pNtpClientProviderConfigData;
```

```
  [case(1)]
    PW32TIME_NTPSERVER_PROVIDER_CONFIG_DATA pNtpServerProviderConfigData;
} W32TIME_PROVIDER_CONFIG_DATA,
 *PW32TIME_PROVIDER_CONFIG_DATA;
```

pNtpClientProviderConfigData: A pointer to a **W32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA** structure that contains configuration data for an NtpClient time provider.

pNtpServerProviderConfigData: A pointer to a **W32TIME_NTPSERVER_PROVIDER_CONFIG_DATA** structure that contains configuration data for an NtpServer time provider.

2.2.11 W32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA

The **W32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA** structure contains configuration data about an NtpClient time provider.

The structure is defined to match the NtpClient time provider's configuration of the W32Time implementation. Fields in the structure that do not apply to other implementations SHOULD have their corresponding configuration-setting type fields set to W32TIME_CONFIGURATION_SETTING_UNDEFINED.<16>

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulAllowNonstandardModeCombinations;
  unsigned __int32 ulCrossSiteSyncFlags;
  unsigned __int32 ulResolvePeerBackoffMinutes;
  unsigned __int32 ulResolvePeerBackoffMaxTimes;
  unsigned __int32 ulCompatibilityFlags;
  unsigned __int32 ulEventLogFlags;
  unsigned __int32 ulLargeSampleSkew;
  unsigned __int32 ulSpecialPollInterval;
  [string, unique] wchar_t* wszType;
  [string, unique] wchar_t* wszNtpServer;
  unsigned __int32 ulAllowNonstandardModeCombinationsFlag;
  unsigned __int32 ulCrossSiteSyncFlagsFlag;
  unsigned __int32 ulResolvePeerBackoffMinutesFlag;
  unsigned __int32 ulResolvePeerBackoffMaxTimesFlag;
  unsigned __int32 ulCompatibilityFlagsFlag;
  unsigned __int32 ulEventLogFlagsFlag;
  unsigned __int32 ulLargeSampleSkewFlag;
  unsigned __int32 ulSpecialPollIntervalFlag;
  unsigned __int32 ulTypeFlag;
  unsigned __int32 ulNtpServerFlag;
  unsigned __int32 cEntries;
  [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA,
 *PW32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA;
```

ulSize: The size, in bytes, of this structure.

ulAllowNonstandardModeCombinations: An integer that indicates whether or not nonstandard mode combinations are allowed.

Value	Meaning
0	MUST indicate that nonstandard mode combinations are not allowed.
1	MUST indicate that nonstandard mode combinations are allowed.

All other values are reserved for future use.

ulCrossSiteSyncFlags: An integer indicating the combination of flags that determines how the time service selects its time source inside a Windows domain.<17> The value MUST be one of the following:

Value	Meaning
NCCSS_None 0x00000000	The time service SHOULD NOT select a time source outside the same site as the machine.
NCCSS_PdcOnly 0x00000001	The time service SHOULD select only the **primary domain controller (PDC)** as its time source.
NCCSS_All 0x00000002	The time service is allowed to select a time source inside or outside the same site as the machine.

All other values are reserved for future use.

ulResolvePeerBackoffMinutes: An integer that indicates the initial time interval, in minutes, to wait after a failure before starting a new time source selection process. The time source selection process is implementation-specific.

ulResolvePeerBackoffMaxTimes: An integer that indicates the maximum number of times to double the wait-time interval when repeated attempts to select a time source fail.

ulCompatibilityFlags: An integer that indicates the combination of flags that determines how the time service validates and accepts a response from its time source. The value MUST be a bitwise OR of zero or more of the following flags.

Value	Meaning
DISPERSION_INVALID 0x00000001	Ignore invalid dispersion check.
IGNORE_FUTURE_REFTIMESTAMP 0x00000002	Ignore reference time-stamp check.
AUTODETECT_WIN2K 0x80000000	Ignore the root dispersion.
AUTODETECT_WIN2K_STAGE2 0x40000000	Ignore the root dispersion by resending a request with a root dispersion of 0xAAAAAAAA.

All other values are reserved for future use.

ulEventLogFlags: An integer that indicates the combination of flags that determines how the time provider logs events into the Windows event log. The value MUST be a bitwise OR of zero or more of the following flags.

Value	Meaning
NCELF_LogReachabilityChanges 0x00000001	Log an event when reachability for a time source has changed.
NCELF_LogIfSampleHasLargeSkew 0x00000002	Log an event when a large time sample skew is detected.
NCELF_LogClientRequestError 0x00000004	Log an event when the time provider fails to receive its response.

All other values are reserved for future use.

ulLargeSampleSkew: An integer that indicates the large time sample skew threshold, in seconds. If a time sample has a difference from the time of the local clock that is more than this value, it MUST be considered to be a sample with large skew.

ulSpecialPollInterval: An integer that indicates a special poll interval, in seconds, for manual time synchronization.

wszType: A case-insensitive, null-terminated string that indicates the time-synchronization behavior of the time service. The string MUST have one of the following values.

Value	Meaning
"NoSync"	The time service does not synchronize with any time source.
"NTP"	The time service synchronizes with a time source specified in **wszNtpServer**. This time synchronization behavior is known as manual time synchronization.
"NT5DS"	The time service synchronizes with a time source inside the Windows domain. The time source is selected using the Windows domain time-source-discovery algorithm. This time-synchronization behavior is known as domain time synchronization.
"AllSync"	The time service synchronizes with both manual time synchronization and domain time synchronization.

All other values are reserved for future use.

wszNtpServer: A case-insensitive, null-terminated string that indicates a space-delimited list of time sources that the time service can synchronize with. The time source SHOULD be represented as either its DNS name or its IP address. An example of this would be "ntp1.nist.gov ntp2.nist.gov ntp3.nist.gov", not including the double-quotes.

ulAllowNonstandardModeCombinationsFlag: An integer that indicates the type of the configuration setting for **ulAllowNonstandardModeCombinations**, as specified in section 2.2.6.

ulCrossSiteSyncFlagsFlag: An integer that indicates the type of the configuration setting for **ulCrossSiteSyncFlags**, as specified in section 2.2.6.

ulResolvePeerBackoffMinutesFlag: An integer that indicates the type of the configuration setting for **ulResolvePeerBackoffMinutes**, as specified in section 2.2.6.

ulResolvePeerBackoffMaxTimesFlag: An integer that indicates the type of the configuration setting for **ulResolvePeerBackoffMaxTimes**, as specified in section 2.2.6.

ulCompatibilityFlagsFlag: An integer that indicates the type of the configuration setting for **ulCompatibilityFlags**, as specified in section 2.2.6.

ulEventLogFlagsFlag: An integer that indicates the type of the configuration setting for **ulEventLogFlags**, as specified in section 2.2.6.

ulLargeSampleSkewFlag: An integer that indicates the type of the configuration setting for **ulLargeSampleSkew**, as specified in section 2.2.6.

ulSpecialPollIntervalFlag: An integer that indicates the type of the configuration setting for **ulSpecialPollInterval**, as specified in section 2.2.6.

ulTypeFlag: An integer that indicates the type of the configuration setting for **wszType**, as specified in section 2.2.6.

ulNtpServerFlag: An integer that indicates the type of the configuration setting for **wszNtpServer**, as specified in section 2.2.6.

cEntries: An integer that indicates the number of additional configuration entries in **pEntries**. The value MUST be set to 0. All other values are reserved for future use.

pEntries: A pointer to **W32TIME_ENTRY** structures that represent additional configuration entries. The value MUST be set to NULL. All other values are reserved for future use.

2.2.12 W32TIME_NTPSERVER_PROVIDER_CONFIG_DATA

The **W32TIME_NTPSERVER_PROVIDER_CONFIG_DATA** structure contains configuration data about an NtpServer time provider.

The structure is defined to match the NtpServer time provider's configuration of the W32Time implementation. Fields in the structure that do not apply to other implementations SHOULD have their corresponding configuration-setting type fields set to W32TIME_CONFIGURATION_SETTING_UNDEFINED.<18>

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulAllowNonstandardModeCombinations;
  unsigned __int32 ulAllowNonstandardModeCombinationsFlag;
  unsigned __int32 ulEventLogFlags;
  unsigned __int32 ulEventLogFlagsFlag;
  unsigned __int32 cEntries;
  [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_NTPSERVER_PROVIDER_CONFIG_DATA,
 *PW32TIME_NTPSERVER_PROVIDER_CONFIG_DATA;
```

ulSize: The size, in bytes, of this structure.

ulAllowNonstandardModeCombinations: An integer that indicates whether or not nonstandard mode combinations are allowed.

Value	Meaning
0	MUST indicate that nonstandard mode combinations are not allowed.
1	MUST indicate that nonstandard mode combinations are allowed.

All other values are reserved for future use.

ulAllowNonstandardModeCombinationsFlag: An integer that indicates the type of the configuration setting for **ulAllowNonstandardModeCombinations**, as specified in section 2.2.6.

ulEventLogFlags: An integer that indicates the combination of flags that determines how the time provider logs events into the Windows event log. This MUST be the following value.

Value	Meaning
NCELF_LogServerResponseError 0x00000008	Log an event when the time provider fails to validate a request for authenticated time synchronization.

All other values are reserved for future use.

ulEventLogFlagsFlag: An integer that indicates the type of the configuration setting for **ulEventLogFlags**, as specified in section 2.2.6.

cEntries: An integer that indicates the number of additional configuration entries in **pEntries**. The value MUST be set to 0. All other values are reserved for future use.

pEntries: A pointer to **W32TIME_ENTRY** structures that represent additional configuration entries. The value MUST be set to NULL. All other values are reserved for future use.

2.2.13 W32TIME_CONFIGURATION_INFO

The **W32TIME_CONFIGURATION_INFO** structure defines the configuration data of the time service.

```
typedef struct {
  unsigned __int32 ulSize;
  W32TIME_CONFIGURATION_BASIC basicConfig;
  W32TIME_CONFIGURATION_ADVANCED advancedConfig;
  W32TIME_CONFIGURATION_DEFAULT defaultConfig;
  unsigned __int32 cProviderConfig;
  PW32TIME_CONFIGURATION_PROVIDER* pProviderConfig;
  unsigned __int32 cEntries;
  [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_CONFIGURATION_INFO,
 *PW32TIME_CONFIGURATION_INFO;
```

ulSize: The size, in bytes, of this structure.

basicConfig: The **W32TIME_CONFIGURATION_BASIC** structure that represents the basic time service configuration data.

advancedConfig: The **W32TIME_CONFIGURATION_ADVANCED** structure that represents the advanced time service configuration data.

defaultConfig: The **W32TIME_CONFIGURATION_DEFAULT** structure that represents the default time service configuration data.

cProviderConfig: The number of time providers that are configured in the time service. This value also indicates the number of structures in **pProviderConfig**.

pProviderConfig: An array of **W32TIME_CONFIGURATION_PROVIDER** structures that represent the configuration data of time providers that are configured in the time service.

cEntries: An integer that indicates the number of additional configuration entries in **pEntries**. The value MUST be set to 0. All other values are reserved for future use.

pEntries: A pointer to **W32TIME_ENTRY** structures that represent additional configuration entries. The value MUST be set to NULL. All other values are reserved for future use.

2.2.14 W32TIME_CONFIGURATION_BASIC

The **W32TIME_CONFIGURATION_BASIC** structure defines the basic configuration data of the time service.

The structure is defined to match the basic configuration of the W32Time implementation. Fields in the structure that are not valid in other implementations SHOULD have their corresponding configuration-setting type fields set to W32TIME_CONFIGURATION_SETTING_UNDEFINED.<19>

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulEventLogFlags;
  unsigned __int32 ulAnnounceFlags;
  unsigned __int32 ulTimeJumpAuditOffset;
  unsigned __int32 ulMinPollInterval;
  unsigned __int32 ulMaxPollInterval;
  unsigned __int32 ulMaxNegPhaseCorrection;
  unsigned __int32 ulMaxPosPhaseCorrection;
  unsigned __int32 ulMaxAllowedPhaseOffset;
  unsigned __int32 ulEventLogFlagsFlag;
  unsigned __int32 ulAnnounceFlagsFlag;
  unsigned __int32 ulTimeJumpAuditOffsetFlag;
  unsigned __int32 ulMinPollIntervalFlag;
  unsigned __int32 ulMaxPollIntervalFlag;
  unsigned __int32 ulMaxNegPhaseCorrectionFlag;
  unsigned __int32 ulMaxPosPhaseCorrectionFlag;
  unsigned __int32 ulMaxAllowedPhaseOffsetFlag;
} W32TIME_CONFIGURATION_BASIC,
 *PW32TIME_CONFIGURATION_BASIC;
```

ulSize: The size, in bytes, of this structure.

ulEventLogFlags: An integer that indicates the combination of flags that determines how the time provider logs events into the Windows event log. The value MUST be a bitwise OR of zero or more of the following flags.

Value	Meaning
EvtLog_TimeJump 0x00000001	Log an event when the time service sets the clock directly to adjust the local clock.
EvtLog_SourceChange 0x00000002	Log an event when the time service synchronizes from a different time source.

All other values are reserved for future use.

ulAnnounceFlags: An integer that indicates the combination of flags that determines how the time service advertises itself as a time server. The value MUST be a bitwise OR of zero or more of the following.

Value	Meaning
Timeserv_Announce_No 0x00000000	Not a time server.
Timeserv_Announce_Yes 0x00000001	Always advertised as a time server.
Timeserv_Announce_Auto 0x00000002	Advertising as a time server is decided automatically: only when the server is synchronized.
Reliable_Timeserv_Announce_Yes 0x00000004	Always advertised as a reliable time server.
Reliable_Timeserv_Announce_Auto 0x00000008	Advertising as a time server is decided automatically: only when the server is synchronized and is a reliable time server.

All other values are reserved for future use.

ulTimeJumpAuditOffset: An integer that indicates the time jump audit threshold, in seconds. If the time service adjusts the local clock by setting the clock directly, and the time correction is more than this value, then the time service logs an audit event.

ulMinPollInterval: An integer that indicates the minimum poll interval of domain time synchronization, expressed as specified in [RFC1305] section 3.2, using units of seconds given as exponents to a power of two.

ulMaxPollInterval: An integer that indicates the maximum poll interval of domain time synchronization, expressed as specified in [RFC1305] section 3.2, using units of seconds given as exponents to a power of two.

ulMaxNegPhaseCorrection: An integer that indicates the maximum negative time correction value, in seconds, that the time service is allowed to use to adjust the local clock. If the time difference is negative and is more than this value, the time service discards the time sample. If the value is set to 0xFFFFFFFF, which is a special case, then the time service MUST be allowed to make any correction.

ulMaxPosPhaseCorrection: An integer that indicates the maximum positive time correction value, in seconds, that the time service is allowed to use to adjust the local clock. If the time difference is positive and is more than this value, the time service discards the time sample. If the value is set to 0xFFFFFFFF, which is a special case, then the time service MUST be allowed to make any correction.

ulMaxAllowedPhaseOffset: An integer that indicates the maximum time correction value, in seconds, that the time service is allowed to use to adjust the local clock, using the clock rate to skew the clock instead of setting the clock directly. If the time difference is more than this value, the time service MUST set the clock directly without skewing.

ulEventLogFlagsFlag: An integer that indicates the type of the configuration setting for **ulEventLogFlags**, as specified in section 2.2.6.

ulAnnounceFlagsFlag: An integer that indicates the type of the configuration setting for **ulAnnounceFlags**, as specified in section 2.2.6.

ulTimeJumpAuditOffsetFlag: An integer that indicates the type of the configuration setting for **ulTimeJumpAuditOffset**, as specified in section 2.2.6.

ulMinPollIntervalFlag: An integer that indicates the type of the configuration setting for **ulMinPollInterval**, as specified in section 2.2.6.

ulMaxPollIntervalFlag: An integer that indicates the type of the configuration setting for **ulMaxPollInterval**, as specified in section 2.2.6.

ulMaxNegPhaseCorrectionFlag: An integer that indicates the type of the configuration setting for **ulMaxNegPhaseCorrection**, as specified in section 2.2.6.

ulMaxPosPhaseCorrectionFlag: An integer that indicates the type of the configuration setting for **ulMaxPosPhaseCorrection**, as specified in section 2.2.6.

ulMaxAllowedPhaseOffsetFlag: An integer that indicates the type of the configuration setting for **ulMaxAllowedPhaseOffset**, as specified in section 2.2.6.

2.2.15 W32TIME_CONFIGURATION_ADVANCED

The **W32TIME_CONFIGURATION_ADVANCED** structure defines the advanced configuration data of the time service.

The structure is defined to match the advanced configuration of the W32Time implementation. Fields in the structure that are not valid in other implementations SHOULD have their corresponding configuration-setting type fields set to W32TIME_CONFIGURATION_SETTING_UNDEFINED.<20>

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 ulFrequencyCorrectRate;
  unsigned __int32 ulPollAdjustFactor;
  unsigned __int32 ulLargePhaseOffset;
  unsigned __int32 ulSpikeWatchPeriod;
  unsigned __int32 ulLocalClockDispersion;
  unsigned __int32 ulHoldPeriod;
  unsigned __int32 ulPhaseCorrectRate;
  unsigned __int32 ulUpdateInterval;
  unsigned __int32 ulFrequencyCorrectRateFlag;
  unsigned __int32 ulPollAdjustFactorFlag;
  unsigned __int32 ulLargePhaseOffsetFlag;
  unsigned __int32 ulSpikeWatchPeriodFlag;
  unsigned __int32 ulLocalClockDispersionFlag;
  unsigned __int32 ulHoldPeriodFlag;
  unsigned __int32 ulPhaseCorrectRateFlag;
  unsigned __int32 ulUpdateIntervalFlag;
} W32TIME_CONFIGURATION_ADVANCED,
```

```
*PW32TIME_CONFIGURATION_ADVANCED;
```

ulSize: The size, in bytes, of this structure.

ulFrequencyCorrectRate: An integer that indicates the correction rate that determines how fast the time service adjusts the clock rate as a frequency correction. For more information on clock discipline algorithms, see [NTP-TR9733i] and [NTP-TR9733] section 4.

ulPollAdjustFactor: An integer that indicates the correction rate that determines how fast the time service adjusts the poll interval. For more information on adjusting the poll interval, see [NTP-TR9733i] and [NTP-TR9733] section 4.

ulLargePhaseOffset: An integer that indicates the threshold that determines whether or not a time sample indicates a spike, in 100 nanosecond units. If the time difference of the time sample is more than the value, the sample MUST indicate a possible spike, in which case the time service MUST change its state, as specified in section 2.2.7. For more information on spike detection, see [NTP-TR9733i] and [NTP-TR9733] section 4.

ulSpikeWatchPeriod: An integer that indicates the time interval, in seconds, that determines how long the time service watches a spike condition. If time samples constantly indicate spikes in this time interval, the time service becomes unsynchronized, in which case the time service MUST change its state, as described in section 2.2.7. For more information on spike detection, see [NTP-TR9733i] and [NTP-TR9733] section 4.3.

ulLocalClockDispersion: An integer that indicates the local clock dispersion, in seconds. The root dispersion is set to this value if the time service runs as a primary server, or if the root dispersion is invalid in the received response. For details on dispersion and root dispersion, see [RFC1305] section 3.2.

ulHoldPeriod: An integer that indicates the number of time samples during which the spike detection is disabled when the time service is in the HOLD state, as specified in section 2.2.7. For more information on the HOLD state, see [NTP-TR9733i] and [NTP-TR9733] section 4.3.

ulPhaseCorrectRate: An integer that indicates the correction rate that determines how fast the time service adjusts the time difference by skewing the clock as a phase correction. For more information on the correction rate, see [NTP-TR9733i] and [NTP-TR9733] section 4.

ulUpdateInterval: An integer that indicates the number of clock ticks between each time correction. For more information on clock ticks, see [NTP-TR9733i] and [NTP-TR9733] section 4.

ulFrequencyCorrectRateFlag: An integer that indicates the type of the configuration setting for **ulFrequencyCorrectRate**, as specified in section 2.2.6.

ulPollAdjustFactorFlag: An integer that indicates the type of the configuration setting for **ulPollAdjustFactor**, as specified in section 2.2.6.

ulLargePhaseOffsetFlag: An integer that indicates the type of the configuration setting for **ulLargePhaseOffset**, as specified in section 2.2.6.

ulSpikeWatchPeriodFlag: An integer that indicates the type of the configuration setting for **ulSpikeWatchPeriod**, as specified in section 2.2.6.

ulLocalClockDispersionFlag: An integer that indicates the type of the configuration setting for **ulLocalClockDispersion**, as specified in section 2.2.6.

ulHoldPeriodFlag: An integer that indicates the type of the configuration setting for **ulHoldPeriod**, as specified in section 2.2.6.

ulPhaseCorrectRateFlag: An integer that indicates the type of the configuration setting for **ulPhaseCorrectRate**, as specified in section 2.2.6.

ulUpdateIntervalFlag: An integer that indicates the type of the configuration setting for **ulUpdateInterval**, as specified in section 2.2.6.

2.2.16 W32TIME_CONFIGURATION_DEFAULT

The **W32TIME_CONFIGURATION_DEFAULT** structure defines the default configuration data of the time service.

The structure is defined to match the default configuration of the W32Time implementation. Fields in the structure that are not valid in other implementations SHOULD have their corresponding configuration-setting type fields set to W32TIME_CONFIGURATION_SETTING_UNDEFINED.<21>

```
typedef struct {
  unsigned __int32 ulSize;
  [string, unique] wchar_t* wszFileLogName;
  [string, unique] wchar_t* wszFileLogEntries;
  unsigned __int32 ulFileLogSize;
  unsigned __int32 ulFileLogFlags;
  unsigned __int32 ulFileLogNameFlag;
  unsigned __int32 ulFileLogEntriesFlag;
  unsigned __int32 ulFileLogSizeFlag;
  unsigned __int32 ulFileLogFlagsFlag;
} W32TIME_CONFIGURATION_DEFAULT,
 *PW32TIME_CONFIGURATION_DEFAULT;
```

ulSize: The size, in bytes, of this structure.

wszFileLogName: A null-terminated string that represents the location of the time service logging file. The location is represented as the full path of the logging file name in the file system. If time service logging is disabled, this value SHOULD be set to a null-terminated empty string.

wszFileLogEntries: A null-terminated string that represents the control that determines what information is logged in the time service logging. The control is implementation-specific.<22>

ulFileLogSize: An integer that indicates the maximum size, in bytes, of the data that the time service can enter into a logging file. If the maximum value is reached, the time service logging MUST wrap around and begin overwriting from the beginning of the file.

ulFileLogFlags: An integer that indicates the time-stamp format of the time service logging. The value MUST be one of the following.

Value	Meaning
FL_HumanReadableTimestamps 0x00000000	Use human-readable time format.
FL_NTTimeEpochTimestamps	Use Windows NT time format.

Value	Meaning
0x00000001	
FL_LocalSystemTimestamps 0x00000002	Use local system format.

All other values are reserved for future use.

ulFileLogNameFlag: An integer that indicates the type of the configuration setting for **wszFileLogName**, as specified in section 2.2.6.

ulFileLogEntriesFlag: An integer that indicates the type of the configuration setting for **wszFileLogEntries**, as specified in section 2.2.6.

ulFileLogSizeFlag: An integer that indicates the type of the configuration setting for **ulFileLogSize**, as specified in section 2.2.6.

ulFileLogFlagsFlag: An integer that indicates the type of the configuration setting for **ulFileLogFlags**, as specified in section 2.2.6.

2.2.17 W32TIME_STATUS_INFO

The **W32TIME_STATUS_INFO** structure defines the current status data of the time service.

```
typedef struct {
  unsigned __int32 ulSize;
  unsigned __int32 eLeapIndicator;
  unsigned __int32 nStratum;
  signed __int32 nPollInterval;
  unsigned __int32 refidSource;
  unsigned __int64 qwLastSyncTicks;
  signed __int64 toRootDelay;
  unsigned __int64 tpRootDispersion;
  signed __int32 nClockPrecision;
  [string, unique] wchar_t* wszSource;
  signed __int64 toSysPhaseOffset;
  unsigned __int32 ulLcState;
  unsigned __int32 ulTSFlags;
  unsigned __int32 ulClockRate;
  unsigned __int32 ulNetlogonServiceBits;
  unsigned __int32 eLastSyncResult;
  unsigned __int64 tpTimeLastGoodSync;
  unsigned __int32 cEntries;
  [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_STATUS_INFO,
 *PW32TIME_STATUS_INFO;
```

ulSize: The size, in bytes, of this structure.

eLeapIndicator: An integer that warns of an impending leap second in the last minute of the current day, as specified in [RFC1305] section 3.2.

nStratum: An integer that indicates the stratum level of the local clock in the time service, as specified in [RFC1305] section 3.2.

nPollInterval: An integer that indicates the poll interval of the time service, expressed as specified in [RFC1305] section 3.2, using units of seconds given as exponents to a power of two.

refidSource: A 32-bit code that identifies the particular reference clock of the time source that the time service is synchronizing with, as specified in [RFC1305] section 3.2.

qwLastSyncTicks: A 64-bit unsigned integer that indicates the most recent time stamp for a successful time synchronization, in 100 nanosecond units.

toRootDelay: A 64-bit signed integer that indicates the total round-trip delay to the primary time source, as specified in [RFC1305] section 3.2, in 100 nanosecond units.

tpRootDispersion: A 64-bit unsigned integer that indicates the root dispersion, as specified in [RFC1305] section 3.2, in 100 nanosecond units.

nClockPrecision: An integer that indicates the time resolution of the local system clock, expressed in the same format as poll intervals that are specified in [RFC1305] section 3.2, using units of seconds given as exponents to a power of two.

wszSource: A null-terminated string that indicates the name of the time source that the time service is synchronizing with. This string SHOULD be either the DNS name or the IP address of the time source in the form of a string, for example, "ntp1.nist.gov" or "10.0.0.1".

toSysPhaseOffset: A 64-bit signed integer that indicates the current time difference between the local clock of the time service and the clock of its time source, in 100 nanosecond units.

ulLcState: An integer that indicates the state of the time service, as specified in section 2.2.7.

ulTSFlags: An integer that indicates the flag combination that represents the properties of the time source. The value MUST be a bitwise OR of zero or more of the following.

Value	Meaning
TSF_Hardware 0x00000001	The time source is from a hardware time provider.
TSF_Authenticated 0x00000002	Authenticated time synchronization is used.
TSF_IPv6 0x00000004	The time source is synchronized via an IP6 address.

All other values are reserved for future use.

ulClockRate: An integer that indicates the current clock rate of the local clock, in units of ticks per second.<23>

ulNetlogonServiceBits: An unsigned 32-bit integer that contains information about the functionality that the time service provides, as specified in section 3.2.5.2.

eLastSyncResult: An integer that indicates the TimeSync_ReturnResult code, as specified in section 3.2.5.1.

tpTimeLastGoodSync: An unsigned 64-bit integer that indicates the time since the time service last performed a successful time synchronization, in 100 nanosecond units.

cEntries: The number of additional configuration entries in **pEntries**. The value MUST be set to 0. All other values are reserved for future use.

pEntries: A pointer to **W32TIME_ENTRY** structures that represent additional configuration entries. The value MUST be set to NULL. All other values are reserved for future use.

2.2.18 W32TIME_ENTRY

The **W32TIME_ENTRY** structure defines the general entry as a possible extension to other time service data structures.<24>

```
typedef struct {
  unsigned __int32 ulSize;
  [string, unique] wchar_t* wszName;
  [string, unique] wchar_t* wszValue;
  [string, unique] wchar_t* wszHelp;
} W32TIME_ENTRY,
 *PW32TIME_ENTRY;
```

ulSize: The size, in bytes, of this structure.

wszName: A null-terminated string that indicates the name of the entry.

wszValue: A null-terminated string that indicates the value of the entry.

wszHelp: A null-terminated string that indicates the display text of the entry.

3 Protocol Details

The following sections specify details of the W32Time Remote Protocol, including abstract data models, interface method syntax, and message processing rules.

3.1 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 that RPC uses internally to implement resiliency to network outages, as specified in [MS-RPCE].

3.1.3 Initialization

The client creates an RPC binding handle, as specified in [C706], to the server RPC endpoint when an RPC method is called. The client MAY create a separate binding handle for each method invocation, or it MAY reuse a binding handle for multiple invocations. The client SHOULD create an authenticated RPC binding handle for the best possible security. (Note that this requires establishing the binding to the well-known endpoint that supports authentication; for more information on authentication, see section 2.1). If a binding to the authenticated RPC endpoint is unavailable (as specified in [MS-RPCE] section 3.3.2), the client SHOULD create a binding handle to the unauthenticated RPC endpoint (for more information on backward compatibility, see 2.1).<25>

If an authenticated binding handle is established, the client SHOULD attempt to establish the strongest possible authentication. If this attempt fails but the binding remains valid (that is, the remote RPC implementation server is listening on the authenticated endpoint but does not support the appropriate authentication mechanism), the client MAY choose to either ignore the error for compatibility or terminate its attempt to invoke the RPC method.<26>

3.1.4 Higher-Layer Triggered Events

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

The following sections provide details about the method invocations.

3.1.4.1 W32TimeSync

Invoking this method SHOULD cause the service to expire the timer that controls the frequency with which the service requests a new time sample from the time source. This will cause the service to immediately request a new time sample from the time source.<27><28><29>

3.1.4.2 W32TimeGetNetlogonServiceBits

No client-specific events or rules are required.

3.1.4.3 W32TimeQueryProviderStatus

<30>

3.1.4.4 W32TimeQuerySource

No client-specific events or rules are required.

3.1.4.5 W32TimeQueryProviderConfiguration

No client-specific events or rules are required.

3.1.4.6 W32TimeQueryConfiguration

No client-specific events or rules are required.

3.1.4.7 W32TimeQueryStatus

No client-specific events or rules are required.

3.1.4.8 W32TimeLog

The client SHOULD change the logging configuration of the time service before invoking this method in order for the time service to update. Otherwise, the same logging configuration is used without update.<31>

3.1.5 Message Processing Events and Sequencing Rules

The client SHOULD ignore errors that the RPC server returns and notify the application invoker of the error received in the higher layer. The client SHOULD ignore values that the RPC server returns that are not described or that are described as reserved for future use, as specified in section 2, and SHOULD notify the application invoker of invalid values in the higher layer. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.<32>

3.1.6 Timer Events

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

3.1.7 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC protocol.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The time service implementing this RPC interface MUST maintain the following abstract data elements that describe its capabilities:

IsTimeServer: A Boolean flag that indicates whether the time service provides a time source using NTP, as specified in [RFC1305].

IsReliableTimeServer: A Boolean flag that indicates whether the time service provides a **reliable time source** using NTP, as specified in [RFC1305].

TimeProviderList: A list of the time providers for the time service. For more information on requirements for each time provider, see section 2.2.4.

3.2.2 Timers

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

3.2.3 Initialization

The server SHOULD<33> listen on both of the well-known endpoints defined for this RPC interface. For more information on endpoints, see section 2.1.

3.2.4 Higher-Layer Triggered Events

No higher-layer triggered events are used.

3.2.5 Message Processing Events and Sequencing Rules

For authenticated RPC over SMB, the details of method authentication are specific to the underlying RPC implementation, as specified in [C706].<34>

The server SHOULD send the values described in section 2. The server SHOULD NOT send any other values that are not described, or that are described as reserved for future use in section 2.<35>

This section specifies the methods for this protocol, in addition to their processing rules.

Methods in RPC Opnum Order

Method	Description
W32TimeSync	Requests that the time service immediately initiate an attempt to synchronize its time. Opnum: 0
W32TimeGetNetlogonServiceBits	Returns information about the functionality the time service provides. Opnum: 1
W32TimeQueryProviderStatus	Returns operational information for a specified time provider. Opnum: 2
W32TimeQuerySource	Returns the current time source of the time service. Opnum: 3
W32TimeQueryProviderConfiguration	Returns configuration data for a specific time provider. Opnum: 4
W32TimeQueryConfiguration	Returns the configuration data of the time service.

Method	Description
	Opnum: 5
W32TimeQueryStatus	Returns the service status data of the time service. Opnum: 6
W32TimeLog	Requests that the time service update its logging configuration. Opnum: 7

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol [MS-RPCE].

The RPC methods shown in the following sections are organized in opnum order.

3.2.5.1 W32TimeSync (Opnum 0)

The **W32TimeSync** method is invoked to request that the time service immediately initiate an attempt to synchronize its time. The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeSync(
  [in] handle_t hBinding,
  [in] unsigned long uWait,
  [in] unsigned long ulFlags
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3, sections Interface Definition Language and Stubs.

uWait: Blocking status of the call. The value MUST be one of the following:

Value	Meaning
0	Method MUST return RPC_S_OK without waiting for the outcome of time synchronization. In this case, the final outcome of the attempt is not available to the caller.
Non-zero	Method MUST NOT return until time synchronization is complete.

ulFlags: Time synchronization behaviors.

The following values SHOULD be mutually exclusive. When multiple values are set, the value whose bit is least significant SHOULD take precedence.<36>

Value	Meaning
TimeSyncFlag_SoftResync 0x00000000	The time service MUST synchronize itself with the currently available time samples. It MUST NOT poll the network or hardware time providers for new time data.
TimeSyncFlag_HardResync 0x00000001	The time service MUST discard its old time samples and MUST acquire new samples from the network or hardware time providers.

Value	Meaning
TimeSyncFlag_Rediscover 0x00000004	Identical to the TimeSyncFlag_HardResync flag, except that the time service MUST attempt to discover new network time sources prior to discarding and reacquiring new time samples.
TimeSyncFlag_UpdateAndResync 0x00000008	Identical to the TimeSyncFlag_Rediscover flag, except that prior to attempting to discover new time sources, the time service MUST update its configuration.
TimeSyncFlag_ForceResync 0x00000010	Identical to the TimeSyncFlag_HardResync flag, except that it causes the processing of the next time sample to ignore any phase correction boundaries imposed by the W32Time service.

The following value MAY be joined in a bitwise OR with the preceding values. If *uWait* is set to zero, the following value MUST be ignored.

Value	Meaning
TimeSyncFlag_ReturnResult 0x00000002	Used only for asynchronous calls. If set, the method MUST return one of the following return values.

Return Values: If the TimeSyncFlag_ReturnResult flag is specified, the return value MUST be one of the following specific TimeSync_ReturnResult codes. Otherwise, this method MUST return zero on success, or an implementation-specific nonzero error code on failure.<37>

Return value/code	Description
0x00000000 ResyncResult_Success	Synchronization between the time provider and the caller was successful. For asynchronous calls, this result does not guarantee that the server has acquired a new time sample. It merely states that the synchronization attempt has been successfully initiated.
0x00000001 ResyncResult_NoData	The time service could not obtain a new time sample from the time provider.
0x00000002 ResyncResult_StaleData	The time service received data that was time-stamped earlier than the last good sample.
0x00000003 ResyncResult_ChangeTooBig	The time service received data in which the time difference from the local clock was too large to trust.
0x00000004 ResyncResult_Shutdown	The time service was shutting down.

The time service MUST immediately attempt to synchronize time with its time sources, as specified by the flags provided in the method invocation. For more information on the flags, see the *ulFlags* table in this section.<38>

3.2.5.2 W32TimeGetNetlogonServiceBits (Opnum 1)

The **W32TimeGetNetlogonServiceBits** method returns information about the functionality that the time service provides. The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeGetNetlogonServiceBits(
  [in] handle_t hBinding
```

```
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3, sections Interface Definition Language and Stubs.

Return Values: This method returns an unsigned 32-bit field that contains information about the functionality that the time service provides. Multiple bits can be set in the return value. Any bits not defined as follows MUST be set to zero by servers and ignored by clients.

Return value/code	Description
0x00000040 DS_TIMESERV_FLAG	The time service provides a time source with which clients can synchronize using NTP, as specified in [RFC1305].
0x00000200 DS_GOOD_TIMESERV_FLAG	The time service provides a reliable time source with which clients can synchronize using NTP, as specified in [RFC1305].

The time service MUST construct a return value that comprises a logical OR of the previously specified flags. In the return value, the bit corresponding to the DS_TIMESERV_FLAG MUST be set to the value of the time service's IsTimeServer data element, and the bit corresponding to the DS_GOOD_TIMESERV_FLAG MUST be set to the value of the time service's IsReliableTimeServer data element.<39>

3.2.5.3 W32TimeQueryProviderStatus (Opnum 2)

The **W32TimeQueryProviderStatus** method returns operational information for a specified time provider (either an NTP or a hardware time provider) within the time service's list of time providers.<40> The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeQueryProviderStatus(
  [in] handle_t hRPCBinding,
  [in] unsigned __int32 ulFlags,
  [in, string] wchar_t* pwszProvider,
  [out, ref] PW32TIME_PROVIDER_INFO* pProviderInfo
);
```

hRPCBinding: RPC primitive binding handle as specified in [C706] part 3, sections Interface Definition Language and Stubs.

ulFlags: Reserved. This parameter SHOULD be set to 0 and SHOULD be ignored on receipt.<41>

pwszProvider: Name of the time provider to query. This name is implementation-specific.<42>

pProviderInfo: A pointer that receives a pointer to a **W32TIME_PROVIDER_INFO** structure containing operational information for the time provider.

Return Values: This method MUST return 0 on success; on failure, it returns a nonzero error code. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for protocol purposes.<43>

When **W32TimeQueryProviderStatus** is invoked, the server SHOULD examine its TimeProviderList to locate the time provider corresponding to the name *pwszProvider*. The server SHOULD return the associated time provider information structure, as specified in section 2.2.1.<44>

3.2.5.4 W32TimeQuerySource (Opnum 3)

The **W32TimeQuerySource** method returns the current time source of the time service. The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeQuerySource(
  [in] handle_t hBinding,
  [out, string] wchar_t** pwszSource
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3, sections Interface Definition Language and Stubs.

pwszSource: A pointer to a null-terminated string that is the name of the time source that the time service is synchronizing with. If the time service is not synchronizing with any time source, the string MUST be set to a null-terminated empty string. This string SHOULD be either the DNS name or the IP address of the time source in the form of a string, for example, "ntp1.nist.gov" or "10.0.0.1".

Return Values: This method MUST return 0 on success; on failure, it returns a nonzero error code. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for the purposes of this protocol.<45>

When **W32TimeQuerySource** is invoked, the server SHOULD return the current time source of the time service.<46>

3.2.5.5 W32TimeQueryProviderConfiguration (Opnum 4)

The **W32TimeQueryProviderConfiguration** method returns configuration data for a specific time provider within the time service's list of time providers. The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeQueryProviderConfiguration(
  [in] handle t hBinding,
  [in] unsigned   int32 ulFlags,
  [in, string] wchar t* pwszProvider,
  [out, ref] PW32TIME_CONFIGURATION_PROVIDER* pConfigurationProviderInfo
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3, sections Interface Definition Language and Stubs.

ulFlags: Reserved. This parameter SHOULD be set to 0 and SHOULD be ignored on receipt.<47>

pwszProvider: A null-terminated string that is the name of the time provider to query. This name is implementation-specific.<48>

pConfigurationProviderInfo: A pointer that receives a pointer to a **W32TIME_CONFIGURATION_PROVIDER** structure containing configuration data for the time provider.

Return Values: This method MUST return 0 on success; on failure, it returns a nonzero error code. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for the purposes of this protocol.<49>

The **W32TimeQueryProviderConfiguration** method is optional. It is specific for the W32Time implementation.<50>

When **W32TimeQueryProviderConfiguration** is invoked, the server SHOULD examine its TimeProviderList to locate the time provider corresponding to the name *pwszProvider*. The server SHOULD return the associated time provider configuration data structure, as specified in section 2.2.8.<51>

3.2.5.6 W32TimeQueryConfiguration (Opnum 5)

The **W32TimeQueryConfiguration** method returns the configuration data of the time service. The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeQueryConfiguration(
  [in] handle_t hBinding,
  [out, ref] PW32TIME_CONFIGURATION_INFO* pConfigurationInfo
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3, sections Interface Definition Language and Stubs.

pConfigurationInfo: A pointer that receives a pointer to a **W32TIME_CONFIGURATION_INFO** structure containing configuration data for the time service.

Return Values: This method MUST return 0 on success; on failure, it returns a nonzero error code. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for the purposes of this protocol.<52>

The **W32TimeQueryConfiguration** method is optional. It is specific for the W32Time implementation.<53>

When **W32TimeQueryConfiguration** is invoked, the server returns the associated time service configuration data structure, as specified in section 2.2.13.

3.2.5.7 W32TimeQueryStatus (Opnum 6)

The **W32TimeQueryStatus** method returns the service status data of the time service. The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeQueryStatus(
  [in] handle_t hBinding,
  [out, ref] PW32TIME_STATUS_INFO* pStatusInfo
);
```

hBinding: RPC primitive binding handle as specified in [C706] part 3, sections Interface Definition Language and Stubs.

pStatusInfo: A pointer that receives a pointer to a **W32TIME_STATUS_INFO** structure containing status data for the time service.

Return Values: This method MUST return 0 on success; on failure, it returns a nonzero error code. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for the purposes of this protocol.<54>

The **W32TimeQueryStatus** method is optional. It is specific for the W32Time service implementation.<55>

When **W32TimeQueryStatus** is invoked, the server returns the associated time service status data structure, as specified in section 2.2.17.

3.2.5.8 W32TimeLog (Opnum 7)

The **W32TimeLog** method is invoked to request that the time service update its logging configuration. The logging of the time service is implementation-specific.<56>

The MIDL syntax of this method is specified as follows.

```
unsigned long W32TimeLog(
  [in] handle_t hBinding
);
```

hBinding: RPC primitive binding handle, as specified in [C706] part 3, sections Interface Definition Language and Stubs.

Return Values: This method MUST return 0 on success; on failure, it returns a nonzero error code. The values transmitted in this field are implementation-specific. All nonzero values MUST be treated as equivalent for protocol purposes.<57>

The **W32TimeLog** method is optional. It is specific for the W32Time implementation.<58>

When **W32TimeLog** is invoked, the server SHOULD<59> update its logging behavior based on its implementation-specific logging configuration.

3.2.6 Timer Events

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

3.2.7 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC protocol.

4 Protocol Example

This section provides an example of how the W32Time Remote Protocol is used.

1. The client obtains the name of a server via an out-of-band mechanism. The client establishes a binding handle to the server at the well-known endpoint \\PIPE\W32TIME_ALT and performs the authentication checks, as specified in section 3.1.3. The client then invokes the **W32TimeGetNetlogonServiceBits** method to query the time service status to check whether the time service provides a time source or a reliable time source.

2. The server receives the **W32TimeGetNetlogonServiceBits** method invocation. In this example, it is assumed that the server allows clients to synchronize time with it by using NTP, as specified in [RFC1305], but that the server is not a reliable time source. In this case, the server returns the value 0x00000040 (DS_TIMESERV_FLAG) to the client.

3. The client receives the return value 0x00000040 (DS_TIMESERV_FLAG), which informs the client that the server is a time source, but not a reliable one.

5 Security

The following sections specify security considerations for implementers of the W32Time Remote Protocol.

5.1 Security Considerations for Implementers

Security considerations for both unauthenticated and authenticated RPC used in this protocol are specified in [MS-RPCE].

The client fails over to unauthenticated RPC when authenticated RPC fails for backward compatibility, as specified in section 3.1.3. The unauthenticated RPC is not as secure as authenticated RPC; the client SHOULD either audit or support this automatic failover only when it is explicitly specified.

5.2 Index of Security Parameters

Security Parameter	Section
Authentication service settings	3.1.3

6 Appendix A: Full IDL

```
typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulResolveAttempts;
    unsigned __int64 u64TimeRemaining;
    unsigned __int64 u64LastSuccessfulSync;
    unsigned __int32 ulLastSyncError;
    unsigned __int32 ulLastSyncErrorMsgId;
    unsigned __int32 ulValidDataCounter;
    unsigned __int32 ulAuthTypeMsgId;
    [string, unique] wchar_t* wszUniqueName;
    unsigned char ulMode;
    unsigned char ulStratum;
    unsigned char ulReachability;
    unsigned char ulPeerPollInterval;
    unsigned char ulHostPollInterval;
} W32TIME_NTP_PEER_INFO, *PW32TIME_NTP_PEER_INFO;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulError;
    unsigned __int32 ulErrorMsgId;
    unsigned __int32 cPeerInfo;
    [size_is(cPeerInfo)] W32TIME_NTP_PEER_INFO pPeerInfo[];
} W32TIME_NTP_PROVIDER_DATA, *PW32TIME_NTP_PROVIDER_DATA;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulError;
    unsigned __int32 ulErrorMsgId;
    [string, unique] wchar_t* wszReferenceIdentifier;
} W32TIME_HARDWARE_PROVIDER_DATA, *PW32TIME_HARDWARE_PROVIDER_DATA;

typedef
[switch_type(unsigned __int32)]
union {
    [case(0)] W32TIME_NTP_PROVIDER_DATA* pNtpProviderData;
    [case(1)] W32TIME_HARDWARE_PROVIDER_DATA* pHardwareProviderData;
} W32TIME_PROVIDER_DATA;

typedef struct {
    unsigned __int32 ulProviderType;
    [switch_is(ulProviderType)] W32TIME_PROVIDER_DATA ProviderData;
} W32TIME_PROVIDER_INFO, *PW32TIME_PROVIDER_INFO;

typedef struct {
    unsigned __int32 ulSize;
    [string, unique] wchar_t *wszName;
    [string, unique] wchar_t *wszValue;
    [string, unique] wchar_t *wszHelp;
} W32TIME_ENTRY, *PW32TIME_ENTRY;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulAllowNonstandardModeCombinations;
    unsigned __int32 ulCrossSiteSyncFlags;
```

```
    unsigned __int32 ulResolvePeerBackoffMinutes;
    unsigned __int32 ulResolvePeerBackoffMaxTimes;
    unsigned __int32 ulCompatibilityFlags;
    unsigned __int32 ulEventLogFlags;
    unsigned __int32 ulLargeSampleSkew;
    unsigned __int32 ulSpecialPollInterval;
    [string, unique] wchar_t* wszType;
    [string, unique] wchar_t* wszNtpServer;
    unsigned __int32 ulAllowNonstandardModeCombinationsFlag;
    unsigned __int32 ulCrossSiteSyncFlagsFlag;
    unsigned __int32 ulResolvePeerBackoffMinutesFlag;
    unsigned __int32 ulResolvePeerBackoffMaxTimesFlag;
    unsigned __int32 ulCompatibilityFlagsFlag;
    unsigned __int32 ulEventLogFlagsFlag;
    unsigned __int32 ulLargeSampleSkewFlag;
    unsigned __int32 ulSpecialPollIntervalFlag;
    unsigned __int32 ulTypeFlag;
    unsigned __int32 ulNtpServerFlag;
    unsigned __int32 cEntries;
    [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA,
  *PW32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulAllowNonstandardModeCombinations;
    unsigned __int32 ulAllowNonstandardModeCombinationsFlag;
    unsigned __int32 ulEventLogFlags;
    unsigned __int32 ulEventLogFlagsFlag;
    unsigned __int32 cEntries;
    [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_NTPSERVER_PROVIDER_CONFIG_DATA,
  *PW32TIME_NTPSERVER_PROVIDER_CONFIG_DATA;

typedef
[switch_type(unsigned __int32)]
union {
    [case(0)] PW32TIME_NTPCLIENT_PROVIDER_CONFIG_DATA
        pNtpClientProviderConfigData;
    [case(1)] PW32TIME_NTPSERVER_PROVIDER_CONFIG_DATA
        pNtpServerProviderConfigData;
} W32TIME_PROVIDER_CONFIG_DATA, *PW32TIME_PROVIDER_CONFIG_DATA;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulProviderType;
    [switch_is(ulProviderType)]
        PW32TIME_PROVIDER_CONFIG_DATA pProviderConfigData;
} W32TIME_PROVIDER_CONFIG, *PW32TIME_PROVIDER_CONFIG;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulEventLogFlags;
    unsigned __int32 ulAnnounceFlags;
    unsigned __int32 ulTimeJumpAuditOffset;
    unsigned __int32 ulMinPollInterval;
    unsigned __int32 ulMaxPollInterval;
    unsigned __int32 ulMaxNegPhaseCorrection;
```

```
    unsigned __int32 ulMaxPosPhaseCorrection;
    unsigned __int32 ulMaxAllowedPhaseOffset;
    unsigned __int32 ulEventLogFlagsFlag;
    unsigned __int32 ulAnnounceFlagsFlag;
    unsigned __int32 ulTimeJumpAuditOffsetFlag;
    unsigned __int32 ulMinPollIntervalFlag;
    unsigned __int32 ulMaxPollIntervalFlag;
    unsigned __int32 ulMaxNegPhaseCorrectionFlag;
    unsigned __int32 ulMaxPosPhaseCorrectionFlag;
    unsigned __int32 ulMaxAllowedPhaseOffsetFlag;
} W32TIME_CONFIGURATION_BASIC, *PW32TIME_CONFIGURATION_BASIC;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulFrequencyCorrectRate;
    unsigned __int32 ulPollAdjustFactor;
    unsigned __int32 ulLargePhaseOffset;
    unsigned __int32 ulSpikeWatchPeriod;
    unsigned __int32 ulLocalClockDispersion;
    unsigned __int32 ulHoldPeriod;
    unsigned __int32 ulPhaseCorrectRate;
    unsigned __int32 ulUpdateInterval;
    unsigned __int32 ulFrequencyCorrectRateFlag;
    unsigned __int32 ulPollAdjustFactorFlag;
    unsigned __int32 ulLargePhaseOffsetFlag;
    unsigned __int32 ulSpikeWatchPeriodFlag;
    unsigned __int32 ulLocalClockDispersionFlag;
    unsigned __int32 ulHoldPeriodFlag;
    unsigned __int32 ulPhaseCorrectRateFlag;
    unsigned __int32 ulUpdateIntervalFlag;
} W32TIME_CONFIGURATION_ADVANCED, *PW32TIME_CONFIGURATION_ADVANCED;

typedef struct {
    unsigned __int32 ulSize;
    [string, unique] wchar_t* wszFileLogName;
    [string, unique] wchar_t* wszFileLogEntries;
    unsigned __int32 ulFileLogSize;
    unsigned __int32 ulFileLogFlags;
    unsigned __int32 ulFileLogNameFlag;
    unsigned __int32 ulFileLogEntriesFlag;
    unsigned __int32 ulFileLogSizeFlag;
    unsigned __int32 ulFileLogFlagsFlag;
} W32TIME_CONFIGURATION_DEFAULT, *PW32TIME_CONFIGURATION_DEFAULT;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 ulInputProvider;
    unsigned __int32 ulEnabled;
    [string, unique] wchar_t* wszDllName;
    [string, unique] wchar_t* wszProviderName;
    unsigned __int32 ulDllNameFlag;
    unsigned __int32 ulProviderNameFlag;
    unsigned __int32 ulInputProviderFlag;
    unsigned __int32 ulEnabledFlag;
    PW32TIME_PROVIDER_CONFIG pProviderConfig;
} W32TIME_CONFIGURATION_PROVIDER, *PW32TIME_CONFIGURATION_PROVIDER;

typedef struct {
```

```
    unsigned __int32 ulSize;
    W32TIME_CONFIGURATION_BASIC basicConfig;
    W32TIME_CONFIGURATION_ADVANCED advancedConfig;
    W32TIME_CONFIGURATION_DEFAULT defaultConfig;
    unsigned __int32 cProviderConfig;
    PW32TIME_CONFIGURATION_PROVIDER* pProviderConfig;
    unsigned __int32 cEntries;
    [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_CONFIGURATION_INFO, *PW32TIME_CONFIGURATION_INFO;

typedef struct {
    unsigned __int32 ulSize;
    unsigned __int32 eLeapIndicator;
    unsigned __int32 nStratum;
    signed __int32 nPollInterval;
    unsigned __int32 refidSource;
    unsigned __int64 qwLastSyncTicks;
    signed __int64 toRootDelay;
    unsigned __int64 tpRootDispersion;
    signed __int32 nClockPrecision;
    [string, unique] wchar_t *wszSource;
    signed __int64 toSysPhaseOffset;
    unsigned __int32 ulLcState;
    unsigned __int32 ulTSFlags;
    unsigned __int32 ulClockRate;
    unsigned __int32 ulNetlogonServiceBits;
    unsigned __int32 eLastSyncResult;
    unsigned __int64 tpTimeLastGoodSync;
    unsigned __int32 cEntries;
    [size_is(cEntries)] PW32TIME_ENTRY pEntries;
} W32TIME_STATUS_INFO, *PW32TIME_STATUS_INFO;


[
    uuid(8fb6d884-2388-11d0-8c35-00c04fda2795),
    version(4.1),
    pointer_default(unique)
]
interface W32Time
{

// Opnum 0
unsigned long
W32TimeSync(
    [in] handle_t hBinding,
    [in] unsigned long uWait,
    [in] unsigned long ulFlags
    );

// Opnum 1
unsigned long
W32TimeGetNetlogonServiceBits(
    [in] handle_t hBinding
    );

// Opnum 2
unsigned long
W32TimeQueryProviderStatus(
```

```
    [in] handle_t hRPCBinding,
    [in] unsigned __int32 ulFlags,
    [in, string] wchar_t *pwszProvider,
    [out, ref] PW32TIME_PROVIDER_INFO* pProviderInfo
    );

// Opnum 3
unsigned long
W32TimeQuerySource(
    [in] handle_t hBinding,
    [out, string] wchar_t** pwszSource
    );

// Opnum 4
unsigned long
W32TimeQueryProviderConfiguration(
    [in] handle_t hBinding,
    [in] unsigned __int32 ulFlags,
    [in, string] wchar_t* pwszProvider,
    [out, ref] PW32TIME_CONFIGURATION_PROVIDER*
        pConfigurationProviderInfo
    );

// Opnum 5
unsigned long
W32TimeQueryConfiguration(
    [in] handle_t hBinding,
    [out, ref] PW32TIME_CONFIGURATION_INFO* pConfigurationInfo
    );

// Opnum 6
unsigned long
W32TimeQueryStatus(
    [in] handle_t hBinding,
    [out, ref] PW32TIME_STATUS_INFO* pStatusInfo
    );

// Opnum 7
unsigned long
W32TimeLog(
    [in] handle_t hBinding
    );

}
```

7 Appendix B: Windows Behavior

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

- 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 1.3: The Windows Time service (W32Time) implements the server end of the W32Time Remote Protocol. For more information on W32Time, see [WTSREF].

<2> Section 1.7: Windows RPC returns RPC_S_PROCNUM_OUT_OF_RANGE to notify the client that an RPC method is out of range, as specified in [MS-ERREF].

<3> Section 2.1: Windows 2000 supports only the unauthenticated RPC interface.

<4> Section 2.2.1: Windows servers do not send any values other than those listed in the possible values. Windows clients ignore any values other than those listed in the possible values.

<5> Section 2.2.3: This field may take on any Windows error code value, as specified in [MS-ERREF].

<6> Section 2.2.3: The **ulErrorMsgId** value is used as an index into a message table stored as a resource in a DLL, as specified in [MS-GLOS], Message Identifier.

<7> Section 2.2.4: This field may take on any Windows error code value, as specified in [MS-ERREF].

<8> Section 2.2.4: The **ulErrorMsgId** value is used as an index into a message table stored as a resource in the DLL that implements the time provider, as specified in [MS-GLOS], Message Identifier.

<9> Section 2.2.5: Windows clients ignore this field. Windows servers set it to the correct value.

<10> Section 2.2.5: Windows does not cap **ulResolveAttempts** and, instead, wraps it around to 0.

<11> Section 2.2.5: This field can take on any Windows error code value, as specified in [MS-ERREF].

<12> Section 2.2.5: In Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, the time provider in W32Time defines the following message identifiers indicating possible time synchronization failure reasons. The failure reasons correspond to message errors, as specified in [RFC1305] section 3.4.4, "Packet Procedure".

Value	Meaning
0x0000005C	The peer is unreachable.
0x0000005E	The time sample was rejected because duplicate time stamps were received from this peer.
0x0000005F	The time sample was rejected because the message was received out of order.
0x00000060	The time sample was rejected because the peer is not synchronized, or because reachability was lost in one or both directions. This value might also indicate that the peer incorrectly sent an NTP request instead of an NTP response.
0x00000061	The time sample was rejected because the round-trip delay was too large, as specified in [RFC1305] section 3.4.4, test 4 of the "Packet Procedure."
0x00000062	The time sample was rejected because the packet was not authenticated.
0x00000063	The time sample was rejected because the peer is not synchronized, or it has been too long since the peer's last synchronization, as specified in [RFC1305], section 3.4.4, test 6 of the "Packet Procedure".
0x00000064	The time sample was rejected because the peer stratum is less than the host stratum, as specified in [RFC1305] section 3.4.4, test 7 of the "Packet Procedure".
0x00000065	The time sample was rejected because the packet contains unreasonable root delay or root dispersion values, as specified in [RFC1305] section 3.4.4, test 8 of the "Packet Procedure".

<13> Section 2.2.5: In Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, the time provider in the W32Time defines the following message identifiers indicating possible authentication mechanisms.

Value	Meaning
0x0000005A	NoAuth: NTP without authentication extension
0x0000005B	NtDigest: NTP with authentication extension

<14> Section 2.2.5: This field is transparently passed through to the invoking application. Windows takes no explicit action on it.

<15> Section 2.2.7: In Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, W32Time implements the clock discipline algorithm. For more information on clock discipline algorithms, see [NTP-TR9733i] and [NTP-TR9733]. The clock discipline algorithm uses a phase lock loop (PLL) method to adjust the local clock's time difference and clock rate as phase correction and frequency correction, and uses a state machine to model the different states of the time adjustment. W32Time supports all the values specified in the possible value table. For more information on clock state machines, see [NTP-TR9733i] and [NTP-TR9733].

<16> Section 2.2.11: The NtpClient time provider's configuration of W32Time is implemented in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP. For more information on W32Time, see [WTSREF].

<17> Section 2.2.11: For more information on domain time source discovery, see [MS-SNTP] and [WTSREF].

<18> Section 2.2.12: The NtpServer time provider's configuration of W32Time is implemented in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, as specified in [WTSREF].

<19> Section 2.2.14: The basic configuration of W32Time is implemented in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, as specified in [WTSREF].

<20> Section 2.2.15: The advanced configuration of W32Time is implemented in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP. For more information on W32Time, see [WTSREF].

<21> Section 2.2.16: The default configuration of W32Time is implemented in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, as specified in [WTSREF].

<22> Section 2.2.16: W32Time uses a string as the control. The string is a list of comma-delimited numbers, each of which indicates the specific information to be logged. Valid numbers are from 0 to 300. Numbers can also be represented as a range, such as 0–300.

<23> Section 2.2.17: The Windows clock rate depends on the computer hardware and Windows releases. There are two common values: 100 ticks per second, where each tick is 10 milliseconds, and 64 ticks per second, where each tick is 15.625 milliseconds.

<24> Section 2.2.18: W32Time does not yet have an extension that uses the **W32TIME_ENTRY** structure.

<25> Section 3.1.3: Windows clients create a separate binding handle for every method invocation. The Windows 2000 client supports only RPC binding via unauthenticated RPC, and only attempts to communicate with the unauthenticated well-known endpoint (see section 2.1). The clients in Windows XP and Windows Server 2003 attempt an RPC binding to the authenticated RPC endpoint first. If the authenticated RPC is unavailable (as specified in [MS-RPCE] section 3.3.2), these clients perform an RPC binding to unauthenticated RPC for backward compatibility with the W32Time implementation on Windows 2000.

Clients determine whether an authenticated RPC binding is available upon method invocation. The Windows implementation creates the binding handle, verifies the security capability of the remote server, and then invokes the method. If the method call fails with RPC_S_SERVER_UNAVAILABLE or RPC_S_UNKNOWN_IF (as specified in [MS-ERREF]), the client attempts to re-establish a binding to the unauthenticated well-known endpoint and retries the method invocation.

<26> Section 3.1.3: For authenticated RPC, the Windows client implementation in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP invokes the rpc__mgmt_inq_princ_name method of the Remote Management Interface, as specified in [C706], and more specifically in the Remote Management Interface Appendix (as specified in [C706] and augmented in [MS-RPCE]) to retrieve the princ_name for the Simple and Protected GSS-API Negotiation (as specified in [MS-SPNG]) authentication service. This invocation is done prior to each W32Time Remote Protocol method call. If this invocation succeeds, authentication with the remote peer is deemed to be possible, and the RPC runtime is configured to use the SPNEGO security provider (with the RPC_C_AUTHN_GSS_NEGOTIATE and RPC_C_AUTHN_LEVEL_PKT_PRIVACY flags), and the retrieved princ_name for the subsequent RPC method calls to the server. If this invocation fails, the Windows client implementation makes all the RPC method calls without authentication.

W32Time Remote Protocol clients in Windows 2000 do not attempt to negotiate authentication.

<27> Section 3.1.4.1: In Windows releases prior to Windows XP, all applications and implementations available as part of Windows set the *ulFlags* parameter to zero.

<28> Section 3.1.4.1: In the default configuration of a non–domain-joined Windows NTP client, the **W32TimeSync** method results in the higher-layer–triggered use of NTP (as described in [RFC1305]). This occurs because a non–domain-joined Windows NTP client is configured to use NTP by default. For more information on NTP, see [RFC1305].

<29> Section 3.1.4.1: In the default configuration of a domain-joined Windows NTP client, the **W32TimeSync** method results in the higher-layer–triggered use of NTP with authentication extensions (as described in [MS-SNTP]). This occurs because a domain-joined Windows NTP client is configured to use NTP Authentication Extensions by default, as outlined in [MS-SNTP] section 1.6. For more information on the Windows implementation of NTP with authentication extensions, see [MS-SNTP].

<30> Section 3.1.4.3: All applications and implementations available as part of Windows set the *ulFlags* parameter to zero.

<31> Section 3.1.4.8: The RPC method **W32TimeLog** is supported in Windows Server 2008 and Windows Vista. The Windows implementation sets up the logging configuration before invoking this method.

<32> Section 3.1.5: The Windows implementation ignores errors and passes them back to the invoker. The Windows implementation does not check returned values, and does not notify the invoker of invalid values. This protocol requests that the RPC engine perform a strict **network data representation (NDR)** data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.1.1.5.3.

<33> Section 3.2.3: The W32Time service in Windows 2000 supports only unauthenticated RPC over SMB, and thus only listens on the well-known endpoint \\PIPE\W32TIME. The W32Time service in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP supports only authenticated RPC over SMB, and thus only listens on \\PIPE\W32TIME_ALT. The W32Time service in Windows XP and Windows Server 2003 does not register any specific authentication mechanism, but rather passes RPC_C_AUTHN_GSS_NEGOTIATE to RPC and allows RPC to negotiate security on behalf of the client and server, as specified in [MS-RPCE].

<34> Section 3.2.5: This protocol requires the RPC runtime to perform a strict NDR data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.1.1.5.3.

For authenticated RPC in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, the W32Time service performs an additional privilege check for all the methods in the RPC interface. The W32Time service checks the client's privileges to determine whether the client possesses the privilege to change system time. If the service is unable to perform this check, or if the client does not possess the required privilege, the RPC method fails. By default, members of the built-in Administrators group possess the privilege to change the system time. For details on RPC security callback, see [MS-RPCE].

<35> Section 3.2.5: Windows servers do not send any values other than those described in section 2.

<36> Section 3.2.5.1: W32Time servers running versions of Windows earlier than Windows XP ignore this field. Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP give precedence to the least significant bit set.

<37> Section 3.2.5.1: If TimeSyncFlag_ReturnResult is not set, this method returns zero on success or any Windows error code value on failure, as specified in [MS-ERREF].

<38> Section 3.2.5.1: The W32Time service in Windows releases earlier than Windows XP ignores the *ulFlags* parameter. The semantics of all invocations of **W32TimeSync** in these releases is that of the TimeSyncFlag_HardResync flag. These releases do not support TimeSyncFlag_ReturnResult

and always return zero for this method. The W32Time service in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP supports the *ulFlags* parameter. When mutually exclusive values are set in the *ulFlags* parameter, as specified in section 3.2.5.1, the W32Time service gives precedence to the flag whose bit is least significant.

<39> Section 3.2.5.2: In Windows releases prior to Windows XP, the W32Time service always sets the DS_TIMESERV_FLAG bit. In Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, the W32Time service sets the DS_TIMESERV_FLAG bit only if it allows clients to synchronize with it using NTP, as specified in [RFC1305].

<40> Section 3.2.5.3: The RPC method **W32TimeQueryProviderStatus** is supported only in Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and service packs. In Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, the W32Time service implements two NTP time providers based on NTP, as specified in [RFC1305]. The well-known time provider names that may be passed as the *pwszProvider* parameter are NtpClient and NtpServer. For more information on W32Time, see [WTSREF]. Windows has not implemented a hardware time provider.

<41> Section 3.2.5.3: Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP set this value to 0 and ignore it on receipt.

<42> Section 3.2.5.3: In Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP, the W32Time service implements two NTP time providers based on NTP, as specified in [RFC1305]. The well-known time provider names that may be passed as the *pwszProvider* parameter are NtpClient and NtpServer. For more information on W32Time, see [WTSREF]. Windows has not implemented a hardware time provider.

<43> Section 3.2.5.3: This field may take on any Windows error code value, as specified in [MS-ERREF].

<44> Section 3.2.5.3: The Windows implementation returns Windows error code ERROR_NOT_FOUND if the time provider is not located.

The W32Time service in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP ignores the *ulFlags* parameter of this method. When constructing the **W32TIME_NTP_PROVIDER_DATA** structure, the time provider that the W32Time service implements always sets the **ulError** and **ulErrorMsgId** fields to zero.

<45> Section 3.2.5.4: This field can take on any Windows error code value, as specified in [MS-ERREF].

<46> Section 3.2.5.4: The RPC method **W32TimeQuerySource** is supported in Windows Server 2008 and Windows Vista. If the time service synchronizes with a time source, the time source name is either a DNS name or an IP address. If the time service does not synchronize with a time source, the name of the time source is set to "Local CMOS Clock".

<47> Section 3.2.5.5: Windows Server 2008 and Windows Vista set this value to 0, and ignore it on receipt.

<48> Section 3.2.5.5: Time providers that the W32Time service supports are described in section 3.2.5.3.

<49> Section 3.2.5.5: This field may take on any Windows error code value, as specified in [MS-ERREF].

<50> Section 3.2.5.5: The RPC method **W32TimeQueryProviderConfiguration** is supported only in Windows Server 2008 and Windows Vista.

<51> Section 3.2.5.5: The Windows implementation returns Windows error code ERROR_NOT_FOUND if the time provider is not located.

<52> Section 3.2.5.6: This field may take on any Windows error code value, as specified in [MS-ERREF].

<53> Section 3.2.5.6: The RPC method **W32TimeQueryConfiguration** is supported only in Windows Server 2008 and Windows Vista.

<54> Section 3.2.5.7: This field may take on any Windows error code value, as specified in [MS-ERREF].

<55> Section 3.2.5.7: The RPC method **W32TimeQueryStatus** is supported only in Windows Server 2008 and Windows Vista.

<56> Section 3.2.5.8: The W32Time service performs logging by writing to a text file stored locally on the machine. Logging is done in a circular fashion, meaning that the logging wraps to the top of the file when it reaches the limit. The size, location, and specific entries to log are configured through the registry.

<57> Section 3.2.5.8: This field may take on any Windows error code value, as specified in [MS-ERREF].

<58> Section 3.2.5.8: The RPC method **W32TimeLog** is supported only in Windows Server 2008 and Windows Vista.

<59> Section 3.2.5.8: W32Time updates its logging behavior based on the logging configuration in the Windows registry. For more information on W32Time, see [WTSREF].

8 Index

T

V

W