# CIFS Remote Administration Protocol

*Preliminary Draft*

## STATUS OF THIS MEMO

## ABSTRACT

This specification defines how an RPC like mechanism may be implemented using the Common Internet File System (CIFS) Transact SMB. Specific examples are provided of how a CIFS client may request a CIFS server to execute a function. The examples show complete details of the request sent by the CIFS client and the response from the CIFS server.

## Table of Contents

# 1.      Objective

This document details an RPC like mechanism used by CIFS clients to submit requests to CIFS servers and obtain the results of the request back from the server.

For convenience, some sections from the CIFS specification have been reproduced in part within this document. Note that the CIFS specification should be considered to be the authoritative reference, in case of any doubts, rather than this document.

## 2.        Prerequisites and suggested reading

- Familiarity with Common Internet File Systems specification (CIFS)

## 3.        Remote Administration Protocol overview

The Remote Administration Protocol (RAP) is similar to an RPC protocol, in that:
- it is an at-most-once synchronous request-response protocol
- it is a framework that can be used for remotely requesting many different kinds of services
- it is designed   to allow (but not require) the programming interface to the protocol to be that of remotely executed procedure calls – which means that one thinks if the protocol in terms of marshaling and unmarshaling procedure call input and output arguments into messages and reliably transporting the messages to and from the client and server
Each RAP request is characterized by a set of ASCII descriptor strings   that are sufficient to be used to interpretively drive the marshaling and unmarshaling process, if an implementation wanted to use them for that purpose. These descriptor strings are included in each request packet, and make the requests self-describing.

RAP is layered on the CIFS Transact SMB, which provides reliable message delivery, security, and messages larger than the underlying network maximum packet size. When used for RAP, the name field in the Transact SMB is always set to "\PIPE\ LANMAN". The Transact SMB is sent on a session/connection that is established to the remote server using a SessionSetupAndX SMB, and using a TID obtained by doing a TreeConnectAndX SMB to a share named "IPC$".

[Refer to the CIFS specification for complete details on SMBs in general, and the Transact SMB in particular. For convenience, relevant portions from the CIFS specification have been reproduced here in Appendix A. Note that the CIFS specification should be considered the authoritative source of information, rather than Appendix A as far as details on the Transact SMB are concerned.]

The model of a RAP service is that there are a few *parameters* as inputs and outputs to the service, exactly one of which may be a buffer descriptor that indicates the presence of a potentially much larger input or output *data buffer*. An argument may be a scalar, pointer, fixed length small array or struct, or a buffer descriptor. The data buffer consists of *entries* followed by a *heap*. An entry consists of a primary data struct and a sequence of   0 or more auxiliary data structs. An input buffer must contain exactly one entry; an output buffer may contain 0 or more. The heap is where data is stored that is referenced by pointers in the entries. The parameters are described by a *parameter descriptor string*; the primary data struct by a *data descriptor string*; and the auxiliary data structs by an *auxiliary data descriptor string*.

## 4.        Remote Administration Protocol

A RAP service request is sent to the server encapsulated in a Transact request SMB and the server sends back a Transact SMB response. An attribute of the Transact SMB is that it divides the payload of request and response messages into two sections: a *parameters* section and a *data* section. As might be expected from the nomenclature, RAP service parameters are sent in the parameters section of a Transact SMB, and the data buffer in the data section. Therefore, to define a service protocol, it is necessary to define the formats of the parameter and data sections of the Transact request and response.

This is done in two stages. First, a C-like declaration notation is used to define descriptor strings, and then the descriptor strings define the formats of the   parameter and data sections.. Note well: even though the declarations may look like a programming interface, they are not: they are a notation for describing the contents of RAP requests and responses; an implementation on any particular system can use any programming interface to RAP services that is appropriate to that system.

### 4.1        Notation

Parameter descriptor strings are defined using a C-like function declaration; data descriptor and auxiliary data descriptor strings are defined using a C-like structure declaration.

Parameter descriptor strings are defined with the following C-like function declaration syntax:

```
rap-service         = "unsigned short" service-name "(" parameters ");"
service-name        = <upper and lower case alpha and numeric>
```

The return type of the function is always "unsigned short", and represents the status code from the function. The service-name is for documentation purposes.

```
parameters          = parameter [ ";" parameter ]
```

The parameter descriptor string for the service is the concatenation of the descriptor characters for the parameters.

```
parameter           = [ "const" ] param-data-type parameter-name
                       [ "[" size "]" ]
param-data-type     = <from parameter descriptor tables below>
parameter-name      = <upper and lower case alpha and numeric>
size                = <string of ASCII 0-9>
```

The descriptor character for a parameter is determined by looking up the data-type in the tables below for request or response parameter descriptors. The parameter-name is for documentation purposes. If there is a size following the parameter-name, then it is placed in the descriptor string following the descriptor character.

Data and auxiliary data descriptor strings are defined   with the following C-like structure declaration syntax:

```
rap-struct          = "struct" struct-name "{" members "}"
```

The descriptor string for the struct is the concatenation of the descriptor characters for the members. The struct-name is for documentation purposes.

```
members             = member [   ";" member ]
member              = member-data-type member-name [ "[" size "]" ]
member-data-type    = <from data descriptor tables below>
```

The descriptor character for a member is determined by looking up the data-type in the tables below for data descriptors. The member-name is for documentation purposes. If there is a size following the member-name, then it is placed in the descriptor string following the descriptor character.

## 4.2        *Descriptors*

The following section contain tables that specify the descriptor character and the notation for each data type for that data type.

### 4.2.1        Request Parameter Descriptors

| Descriptor | Data Type | Format |
|---|---|---|
| W | **unsigned short** | indicates parameter type of 16 bit integer (word). |
| D | **unsigned long** | indicates parameter type of 32 bit integer (dword). |
| b | **BYTE** | indicates bytes (octets). May be followed by an ASCII number indicating number of bytes.. |
| O | **NULL** | indicates a NULL pointer |
| z | **char** | indicates a NULL terminated ASCII string present in the parameter area |
| F | **PAD** | indicates Pad bytes (octets). May be followed by an ASCII number indicating the number of bytes |
| r | **RCVBUF** | pointer to receive data buffer in response parameter section |
| L | **RCVBUFLEN** | 16 bit integer containing length of receive data buffer in (16 bit) words |
| s | **SNDBUF** | pointer to send data buffer in request parameter section |
| T | **SNDBUFLEN** | 16 bit integer containing length of send data buffer in words |

### 4.2.2          Response Parameter Descriptors

| Descriptor | Data Type | Format |
|===========|=========|=====|
| g | **BYTE \*** | indicates a byte is   to be received. May be followed by an ASCII number indicating number of bytes to receive |
| h | **unsigned short \*** | indicates a word is to be received |
| i | **unsigned long \*** | indicates   a dword is to be received |
| e | **ENTCOUNT** | indicates a word is to be received   which indicates the number of entries returned |

### 4.2.3          Data Descriptors

| Descriptor | Data Type | Format |
|===========|=========|=====|
| W | **unsigned short** | indicates data type of 16 bit integer (word). Descriptor char may be followed by an ASCII number indicating the number of 16 bit words present |
| D | **unsigned long** | indicates data type of 32 bit integer (dword). Descriptor char may be followed by an ASCII number indicating the number of 32 bit words present |
| B | **BYTE** | indicates item of data type 8 bit byte (octet). The indicated number of bytes are present   in the data. Descriptor char may be followed by an ASCII number indicating the number of 8 bit bytes present |
| O | **NULL** | indicates a NULL pointer |
| z | **char \*** | indicates a 32 bit pointer to a NULL terminated ASCII string is present in the response parameter area. The actual string is in the response data area and the pointer in the parameter area points to the string in the data area. The high word of the pointer should be ignored. The converter word present in the response parameter section should be subtracted from the low 16 bit value to obtain an offset into the data area indicating where the data area resides. |
| N | **AUXCOUNT** | indicates number of auxiliary data structures. The transaction response data section contains an unsigned 16 bit number corresponding to this data item. |

### 4.3          *Transaction Request Parameters section*

The parameters and data being sent and received are described by ASCII descriptor strings. These descriptor strings are described in section 4.2.

The parameters section of the Transact SMB request contains the following (in the order described)
•          The function number: an unsigned short 16 bit integer identifying the function being remoted
•          The parameter descriptor string: a null terminated ASCII string
•          The data descriptor string: a null terminated ASCII string.
•          The request parameters, as described by the parameter descriptor string, in the order that the request parameter descriptor characters appear in the parameter descriptor string
•          An optional auxiliary data descriptor string: a null terminated ASCII string. It will be present if there is an auxiliary data structure count in the primary data struct (an "N" descriptor in the data descriptor string).

RAP requires that the length of the return parameters be less than or equal to the length of the parameters being sent; this requirement is made to simply buffer management in implementations. This is reasonable as the functions were designed to return data in the data section and use the return parameters for items like data length, handles, etc. If need be, this restriction can be circumvented by filling in some pad bytes into the parameters being sent.

### 4.4          *Transaction Request Data section*

The Data section for the transaction request is present if the parameter description string contains an "s" (SENDBUF) descriptor. If present, it contains:
•          A primary data struct, as described by the data descriptor string
•          Zero or more instances of the auxiliary data struct, as described by the auxiliary data descriptor string. The number of instances is determined by the value of the an auxiliary data structure count member of the primary data struct, indicated by the "N" (AUXCOUNT) descriptor. The auxiliary data is present only if the auxiliary data descriptor string is non null.
•          Possibly some pad bytes
•          The heap: the data referenced by pointers in the primary and auxiliary data structs.

### 4.5          *Transaction Response Parameters section*

The response sent by the server contains a parameter section which consists of:
•          A 16 bit integer indicating the status or return code. The possible values for different functions are different.
•          A 16 bit converter word, used adjust pointers to information in the response data section. Pointers returned within the response buffer are 32 bit pointers. The high order 16 bit word should be ignored. The converter word needs to be subtracted from the low order   16 bit word to arrive at an offset into the response buffer.
•          The response parameters, as described by the parameter descriptor string, in the order that the response parameter descriptor characters appear in the parameter descriptor string.

### 4.6          *Transaction Response Data section*

The Data section for the transaction response is present if the parameter description string contains an "r" (RCVBUF) descriptor. If present, it contains:
•          Zero or more entries. The number of entries is determined by the value of the entry count parameter, indicated by the "e"(ENTCOUNT) descriptor. Each entry contains:
•          A primary data struct, as described by the data descriptor string
•          Zero or more instances of the auxiliary data struct, as described by the auxiliary data descriptor string. The number of instances is determined by the value of the AUXCOUNT member of the primary data struct (whose descriptor is "N"). The auxiliary data is present only if the auxiliary data descriptor string is non null.
•          Possibly some pad bytes
•          The heap: the data referenced by pointers in the primary and auxiliary data structs.

# 5.        NetShareEnum

The NetShareEnum RAP function retrieves information about each shared resource on a CIFS server. The definition is:

> **unsigned** short **NetShareEnum**(
>
>     **unsigned short**       *sLevel*;
>     **RCVBUF**           *pbBuffer*;
>     **RCVBUFLEN**       *cbBuffer*;
>     **ENTCOUNT**       *pcEntriesRead*;
>     **unsigned short**       **pcTotalAvail*;
> );

where:

> sLevel specifies the level of detail returned and must have the value of 1.

> pbBuffer points to the buffer to receive the returned data. If the function is successful, the buffer contains a sequence of **SHARE_INFO_1** structures (defined later).

> cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

> pcEntriesRead points to a 16 bit variable that receives a count of the number of shared resources enumerated in the buffer. This count is valid only if **NetShareEnum** returns the NERR_Success or ERROR_MORE_DATA values.

> pcTotalAvail points to a 16-bit variable that receives a count of the total number of shared resources. This count is valid only if **NetShareEnum** returns the NERR_Success or ERROR_MORE_DATA values.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

-           The 16 bit function number for NetShareEnum which is 0.
-           The parameter descriptor string which is "WrLeh".
-           The data descriptor string for the (returned) data which is "B13BWz"
-           The actual parameters as described by the parameter descriptor string.

The parameters are:
-           A 16 bit integer with a value of 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return
-           A 16 bit integer that contains the size of the receive buffer.

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
-           A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|------|-------|-------------|
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| ERROR_NETWORK_ACCESS_DENIED | 65 | Network access is denied |
| ERROR_MORE_DATA | 234 | Additional data is available |
| NERR_ServerNotStarted | 2114 | The server service on the remote computer is not running |
| NERR_BadTransactConfig | 2141 | The server is not configured for transactions, IPC$ is not shared |

- A 16 bit "converter" word.
- A 16 bit number representing the number of entries returned.
- A 16 bit number representing the total number of available entries. If the supplied buffer is large enough, this will equal the number of entries returned.

## Transaction Response Data section

The return data section consists of a number of SHARE_INFO_1 structures. The number of such structures present is determined by the third entry (described above) in the return parameters section.

The **SHARE_INFO_1** structure is defined as:

```
struct SHARE_INFO_1 {
    char              shi1_netname[13]
    char              shi1_pad;
    unsigned short    shi1_type
    char              *shi1_remark;
}
```

where:

shi1_netname contains a null terminated ASCII string that specifies the share name of the resource.

shi1_pad aligns the next data strructure element to a word boundary.

shi1_type contains an integer that specifies the type of the shared resource. The possible values are:

| Name | Value | Description |
|------|-------|-------------|
| STYPE_DISKTREE | 0 | Disk Directory Tree |
| STYPE_PRINTQ | 1 | Printer Queue |
| STYPE_DEVICE | 2 | Communications device |
| STYPE_IPC | 3 | Inter process communication (IPC) |

shi1_remark points to a null terminated ASCII string that contains a comment abthe shared resource. The value for shi1_remark is null for ADMIN$ and IPC$ share names.   The shi1_remark pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

In case there are multiple SHARE_INFO_1 data structures to return, the server may put all these fixed length structures in the return buffer, leave some space and then put all the variable length data (the actual value of the shi1_remark strings) at the end of the buffer.

There is no auxiliary data to receive.


# 6.        NetServerEnum2

The **NetServerEnum2** RAP service lists all computers of the specified type or types that are visible in the specified domains. It may also enumerate domains. The definition is:

**unsigned short NetServerEnum2 (**
       **unsigned short**      s*Level*,
       **RCVBUF**         pbBuffer,
       **RCVBUFLEN**     cbBuffer,
       **ENTCOUNT**      pcEntriesRead,

```
        unsigned short        *pcTotalAvail,
        unsigned long         fServerType,
        char                  *pszDomain,
        );
```

where:

sLevel specifies the level of detail (0 or 1) requested.

pbBuffer points to the buffer to receive the returned data. If the function is successful, the buffer contains a sequence of **server_info_**_x_ structures, where _x_ is 0 or 1, depending on the level of detail requested.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcEntriesRead points to a 16 bit variable that receives a count of the number of servers enumerated in the buffer. This count is valid only if **NetServerEnum2** returns the NERR_Success or ERROR_MORE_DATA values.

pcTotal Avail points to a 16 bit variable that receives a count of the total number of available entries. This count is valid only if **NetServerEnum2** returns the NERR_Success or ERROR_MORE_DATA values.

fServerType specifies the type or types of computers to enumerate. Computers that match at least one of the specified types are returned in the buffer. Possible values are defined in the request parameters section.

pszDomain points to a null-terminated string that contains the name of the workgroup in which to enumerate computers of the specified type or types. If the *pszDomain* parameter is a null string or a null pointer, servers are enumerated for the current domain of the computer.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetServerEnum2 which is 104.
- The parameter descriptor string which is "WrLehDz".
- The data descriptor string for the (returned) data which is "B16" for level detail 0 or "B16BBDz" for level detail 1.
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A 16 bit integer with a value of 0 or 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return
- A 16 bit integer that contains the size of the receive buffer.
- A 32 bit integer that represents the type of servers the function should enumerate. The possible values may be any of the following or a combination of the following:

| | | | |
|---|---|---|---|
| SV_TYPE_WORKSTATION | 0x00000001 | All workstations |
| SV_TYPE_SERVER | 0x00000002 | All servers |
| SV_TYPE_SQLSERVER | 0x00000004 | Any server running with SQL server |
| SV_TYPE_DOMAIN_CTRL | 0x00000008 | Primary domain controller |
| SV_TYPE_DOMAIN_BAKCTRL | 0x00000010 | Backup domain controller |
| SV_TYPE_TIME_SOURCE | 0x00000020 | Server running the timesource service |
| SV_TYPE_AFP | 0x00000040 | Apple File Protocol servers |
| SV_TYPE_NOVELL | 0x00000080 | Novell servers |
| SV_TYPE_DOMAIN_MEMBER | 0x00000100 | Domain Member |
| SV_TYPE_PRINTQ_SERVER | 0x00000200 | Server sharing print queue |
| SV_TYPE_DIALIN_SERVER | 0x00000400 | Server running dialin service. |
| SV_TYPE_XENIX_SERVER | 0x00000800 | Xenix server |
| SV_TYPE_NT | 0x00001000 | NT server |
| SV_TYPE_WFW | 0x00002000 | Server running Windows for Workgroups |
| SV_TYPE_SERVER_NT | 0x00008000 | Windows NT non DC server |
| SV_TYPE_POTENTIAL_BROWSER | 0x00010000 | Server that can run the browser service |
| SV_TYPE_BACKUP_BROWSER | 0x00020000 | Backup browser server |
| SV_TYPE_MASTER_BROWSER | 0x00040000 | Master browser server |
| SV_TYPE_DOMAIN_MASTER | 0x00080000 | Domain Master Browser server |
| SV_TYPE_LOCAL_LIST_ONLY | 0x40000000 | Enumerate only entries marked "local" |
| SV_TYPE_DOMAIN_ENUM | 0x80000000 | Enumerate Domains. The pszServer and pszDomain parameters must be NULL. |

- A null terminated ASCII string representing the pszDomain parameter described above

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|---|---|---|
| NERR_Success | 0 | No errors encountered |
| ERROR_MORE_DATA | 234 | Additional data is available |
| NERR_ServerNotStarted | 2114 | The RAP service on the remote computer is not running |
| NERR_BadTransactConfig | 2141 | The server is not configured for transactions, IPC$ is not shared |

- A 16 bit "converter" word.
- A 16 bit number representing the number of entries returned.
- A 16 bit number representing the total number of available entries. If the supplied buffer is large enough, this will equal the number of entries returned.

## Transaction Response Data section

The return data section consists of a number of SHARE_INFO_1 structures. The number of such structures present is determined by the third entry (described above) in the return parameters section.

At level detail 0, the Transaction response data section contains a number of SERVER_INFO_0 data structure. The number of such structures is equal to the 16 bit number returned by the server in the third parameter in the Transaction response parameter section. The SERVER_INFO_0 data structure is defined as:

```
struct SERVER_INFO_0 {
    char     sv0_name[16];
};
```

where:

sv0_name is a null-terminated string that specifies the name of a computer or domain .

At level detail 1, the Transaction response data section contains a number of SERVER_INFO_1 data structure. The number of such structures is equal to the 16 bit number returned by the server in the third parameter in the Transaction response parameter section. The SERVER_INFO_1 data structure is defined as:

```
struct SERVER_INFO_1 {
    char            sv1_name[16];
    char            sv1_version_major;
    char            sv1_version_minor;
    unsigned long   sv1_type;
    char            *sv1_comment_or_master_browser;
};
```

sv1_name contains a null-terminated string that specifies the name of a computer.

sv1_version_major specifies the major release version number of the networking software the server is running. This is entirely informational and something the caller of the NetServerEnum2 function gets to see.

sv1_version_minor specifies the minor release version number of the networking software the server is running. This is entirely informational and something the caller of the NetServerEnum2 function gets to see.

sv1_type specifies the type of software the computer is running. The member can be one or a combination of the values defined above in the Transaction request parameters section for fServerType.

sv1_comment_or_master_browser points to a null-terminated string. If the sv1_type indicates that the entry is for a domain, this specifies the name of the domain master browser; otherwise, it specifies a comment describing the server. The comment can be a null string or the pointer may be a null pointer.

In case there are multiple SERVER_INFO_1 data structures to return, the server may put all these fixed length structures in the return buffer, leave some space and then put all the variable length data (the actual value of the sv1_comment strings) at the end of the buffer.

There is no auxiliary data to receive.

# 7.        NetServerGetInfo

The NetServerGetInfo function returns information about the specified server. The definition is:

```
unsigned short NetServerGetInfo(
    unsigned short      sLevel;
    RCVBUF              pbBuffer;
    RCVBUFLEN           cbBuffer;
    unsigned short      *pcbTotalAvail;
);
```

where:

   sLevel specifies the level of detail returned.   (Legal values are 0 and 1)

   pbBuffer points to the buffer to receive the returned data.

   cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

   pcbTotalAvail points to a   16 bit variable that receives a count of the total number of bytes of information available. This count is valid only if **NetServerGetInfo** returns the NERR_Success or ERROR_MORE_DATA values.

   The return value is one of the following:

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetServerGetInfo which is 13.
- The parameter descriptor string which is "WrLh"
- The data descriptor string for the (returned) data which is "B16" for level detail 0 or   "B16BBDz" for level detail 1.
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A 16 bit integer with a value of 0 or 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return
- A 16 bit integer that contains the size of the receive buffer.

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|---|---|---|
| NERR_Success | 0 | No errors encountered |
| ERROR_MORE_DATA | 234 | Additional data is available |
| NERR_ServerNotStarted | 2114 | The RAP service on the remote computer is not running |
| NERR_BadTransactConfig | 2141 | The server is not configured for transactions, IPC$ is not shared |

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or ERROR_MORE_DATA. In case of success, this will indicate the number of useful bytes available. In case of failure, this indicates the required size of the receive buffer.

**Transaction Response Data section**

At level detail 0, the Transaction response data section contains a SERVER_INFO_0 data structure.   The SERVER_INFO_0 data structure is defined in section 7.4

At level detail 1, the Transaction response data section contains a SERVER_INFO_1 data structure.   The SERVER_INFO_1 data structure is defined in section 7.4

There is no auxiliary data to receive.

# 8.      NetShareGetInfo

The NetShareGetInfo function retrieves information about a particular shared resource on a CIFS server. The definition is:

**unsigned short NetShareGetInfo**(
    **char**                  *\*pszNetName*;
    **unsigned short**     *sLevel*;
    **RCVBUF**                   p*bBuffer*;
    **RCVBUFLEN**              *cbBuffer*;

    **unsigned short**     *\*pcbTotalAvail*;
);

where:

pszNetName points to an ASCII null-terminated string specifying the name of the shared resource for which information should be retrieved.

sLevel specifies the level of detail returned. (Legal values are 0, 1 and 2)

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail points to a 16 bit variable that receives a count of the total number of bytes of information available. This count is valid only if **NetShareGetInfo** returns the NERR_Success or ERROR_MORE_DATA values.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetServerGetInfo which is 1.
- The parameter descriptor string which is "zWrLh"
- The data descriptor string for the (returned) data which is "B13" for level detail 0 or "B13BWz" for level detail 1 or "B13BWzWWWzB9B" for level detail 2.
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A null terminated ASCII string indicating the share for which information should be retrieved.
- A 16 bit integer with a value of 0, 1 or 2 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return
- A 16 bit integer that contains the size of the receive buffer.

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|---|---|---|
| NERR_Success | 0 | No errors encountered |
| ERROR_MORE_DATA | 234 | Additional data is available |
| NERR_ServerNotStarted | 2114 | The RAP service on the remote computer is not running |
| NERR_BadTransactConfig | 2141 | The server is not configured for transactions, IPC$ is not shared |

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or   ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

## Transaction Response Data section

At level detail 0, the Transaction response data section contains a SHARE_INFO_0 data structure, which is defined as:

```
struct SHARE_INFO_0 {
     char           shi1_netname[13]
}
```

 where:

shi0_netname contains an ASCIIZ string that specifies the share name of the resource.

At level detail 1, the Transaction response data section contains a SHARE_INFO_1 data structure, which is defined as:

```
struct SHARE_INFO_1 {
     char           shi1_netname[13]
     char           shi1_pad;
     unsigned short shi1_type
     char             *shi1_remark;
}
```

 where

shi1_netname contains an ASCIIZ string that specifies the share name of the resource.

shi1_pad aligns the next data structure element to a word boundary.

shi1_type contains an integer that specifies the type of the shared resource. The possible values are:

| Name | Value | Description |
|---|---|---|
| STYPE_DISKTREE | 0 | Disk Directory Tree |
| STYPE_PRINTQ | 1 | Printer Queue |
| STYPE_DEVICE | 2 | Communications device |
| STYPE_IPC | 3 | Inter process communication (IPC) |

shi1_remark points to a null-terminated string that specifies a comment describing the share. The comment can be a null string or the pointer may be a null pointer.
The shi1_remark pointer is a 32 bit pointer. The higher 16 bits must be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

At level detail 2, the Transaction response data section contains a SHARE_INFO_2 data structure, which is defined as:

```
struct SHARE_INFO_2 {
      char                shi2_netname[13]
      char                shi2_pad;
      unsigned short      shi2_type
      char    *           shi2_remark;
      unsigned short      shi2_permissions;
      unsigned short      shi2_max_uses;
      unsigned short      shi2_current_uses;
      unsigned short      shi2_path;
      unsigned short      shi2_passwd[9]
      unsigned short      shi2_pad2;
}
```

where

shi2_netname contains a null terminated ASCII string that specifies the share name of the resource.

shi2_pad aligns the next data strructure element to a word boundary.

shi2_type contains an integer that specifies the type of the shared resource. The possible values are:

| Name | Value | Description |
|------|-------|-------------|
| STYPE_DISKTREE | 0 | Disk Directory Tree |
| STYPE_PRINTQ | 1 | Printer Queue |
| STYPE_DEVICE | 2 | Communications device |
| STYPE_IPC | 3 | Inter process communication (IPC) |

shi2_remark is a pointer to a null terminated ASCII string specifying a comment for the share

shi2_permissions specifies the permissions on the shared resource if the CIFS server is operating with share level security. The values are this element can take are defined as a series of bit masks that may be OR'ed with each other. The bit mask values are:

| Name | Bit Mask Value | Description |
|------|----------------|-------------|
| ACCESS_READ | 0x01 | Permission to read & execute from resource |
| ACCESS_WRITE | 0x02 | Permission to write data to resource |
| ACCESS_CREATE | 0x04 | Permission to create an instance of  the resource |
| ACCESS_EXEC | 0x08 | Permission to execute from resource |
| ACCESS_DELETE | 0x10 | Permission to delete the resource |
| ACCESS_ATRIB | 0x20 | Permission to modify the resource attributes such as date & time of last modification, etc |
| ACCESS_PERM | 0x40 | Permission to change permissions on the resource |
| ACCESS_ALL | 0x7F | All of the above permissions |

shi2_max_uses specifies the maximum number of current uses the shared resource can accommodate. A Value of -1 indicates there is no limit.

shi2_current_uses specifies the current number of connections to the resource

shi2_path point to an ASCIIZ string that contains the local (on the remote CIFS server) path   name of the shared resource.
- For printer resources, shi2_path specifies   the name of the printer queue being shared
- For disk devices, shi2_path specifies the path being shared
- For communication device queues, shi2_path specifies the name of the of the communication device
- For ADMIN$ or IPC$ resources, shi2_path must be a null pointer

shi2_passwd specifies the password for the resource in case the CIFS server is running with share level security. For CIFS servers running with user level security, this field is set to null and is ignored.

shi2_pad2 is just a pad byte

All of the pointers to an ASCII string in this data structure (shi2_remark and shi2_path) need to be treated specially. The   pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

There is no auxiliary data in the response.

## 9.        NetwkstaUserLogon

This is a function executed on a remote CIFS server to log on a user. The purpose is to perform checks such as whether the specified user is permitted to logon from the specified computer, whether the specified user is permitted to log on at the given moment, etc. as well as perform housekeeping and statistics updates.

There is a password field in the parameters for this function. However, this field is always set to null before the function is sent on the wire, in order to preserve security. The remote CIFS server ignores this meaningless password that is sent. The remote CIFS server ensures security by checking that the user name and computer name that are in the request parameters are the same used to establish the session and connection to the IPC$ share on the remote CIFS server.

The definition is:

  **unsigned short NetWkstaUserLogon**(

```
char            *reserved1;
char            *reserved2;
unsigned short   sLevel;
BYTE            bReqBuffer[54];
unsigned short  cbReqBuffer;
RCVBUF          pbBuffer;
RCVBUFLEN       cbBuffer;
unsigned short  *pcbTotalAvail;
);
```

where:

reserved1 and reserved2 are reserved fields and must be null.

sLevel specifies the level of detail returned. The only legal value is 1.

pbReqBuffer points to the request buffer. This buffer contains parameters that need to be sent to the server. The actual value and structure is defined in the Transaction Request Parameters section.

cbReqBuffer specifies the size, in bytes, of the buffer pointed to by the *pbReqBuffer* parameter.   The value must be decimal 54.

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetWkstaUserLogon which is 132.
- The parameter descriptor string which is "OOWb54WrLh"

- The data descriptor string for the (returned) data which is "WB21BWDWWDDDDDDDzzzD"
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A 16 bit integer with a value of 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- a byte array of length 54 bytes. These 54 bytes are defined as

```
char wlreq1_name[21];        // User Name
char wlreq1_pad1;            //Pad next field to a word boundary
char wlreq1_password[15];    //Password, set to null, ignored by server
char wlreq1_pad2;            //Pad next field to word boundary
char wlreq1_workstation[16]; //ASCII name of computer
```
- A 16 bit integer with a value of 54
- A 16 bit integer that contains the size of the receive buffer

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|---|---|---|
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| NERR_LogonScriptError | 2212 | An error occurred while loading or running the logon script |
| NERR_StandaloneLogon | 2214 | The logon was not validated by any server |
| NERR_NonValidatedLogon | 2217 | The logon server is running an older software version and cannot validate the logon |
| NERR_InvalidWorkstation | 2240 | The user is not allowed to logon from this computer |
| NERR_InvalidLogonHours | 2241 | The user is not allowed to logon at this time |
| NERR_PasswordExpired | 2242 | The user password has expired |

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or   ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

## Transaction Response Data section

The Transaction response data section contains a data structure user_logon_info_1 which is defined as:

```
struct user_logon_info_1 {
      unsigned short      usrlog1_code;
      char                usrlog1_eff_name[21];
      char                  usrlog1_pad_1;
      unsigned short      usrlog1_priv;
      unsigned long       usrlog1_auth_flags;
      unsigned short      usrlog1_num_logons;
      unsigned short      usrlog1_bad_pw_count;
      unsigned long       usrlog1_last_logon;
      unsigned long       usrlog1_last_logoff;
      unsigned long       usrlog1_logoff_time;
      unsigned long       usrlog1_kickoff_time;
      long                  usrlog1_password_age;
      unsigned long       usrlog1_pw_can_change;
      unsigned long       usrlog1_pw_must_change;
      char                *usrlog1_computer;
      char                *usrlog1_domain;
      char                *usrlog1_script_path;
      unsigned long       usrlog1_reserved1;
};
```

where:

usrlog1_code specifies the result and can have the following values:

| Code | Value | Description |
|---|---|---|
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| NERR_LogonScriptError | 2212 | An error occurred while loading or running the logon script |
| NERR_StandaloneLogon | 2214 | The logon was not validated by any server |
| NERR_NonValidatedLogon | 2217 | The logon server is running an older software version and cannot validate the logon |
| NERR_InvalidWorkstation | 2240 | The user is not allowed to logon from this computer |
| NERR_InvalidLogonHours | 2241 | The user is not allowed to logon at this time |
| NERR_PasswordExpired | 2242 | Administrator privilege |

usrlog1_eff_name specifies the account to which the user was logged on

usrlog1_pad1 aligns the next data structure element to a word boundary

usrlog1_priv specifies the user's privilege level. The possible values are:

| Name | Value | Description |
|------|-------|-------------|
| USER_PRIV_GUEST | 0 | Guest privilege |
| USER_PRIV_USER | 1 | User privilege |
| USER_PRV_ADMIN | 2 | Administrator privilege |

usrlog1_auth_flags specifies the account operator privileges. The possible values are:

| Name | Value | Description |
|------|-------|-------------|
| AF_OP_PRINT | 0 | Print operator |
| AF_OP_COMM | 1 | Communications operator |
| AF_OP_SERVER | 2 | Server operator |
| AF_OP_ACCOUNTS | 3 | Accounts operator |

usrlog1_num_logons specifies the number of times this user has logged on. A value of -1 means the number of logons is unknown.

usrlog1_bad_pw_count specifies the number of incorrect passwords entered since the last successful logon.

usrlog1_last_logon specifies the time when the user last logged on. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970.

usrlog1_last_logoff specifies the time when the user last logged off. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970. A value of 0 means the last logoff   time is unknown.

usrlog1_logoff_time specifies the time when the user should logoff. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970. A value of -1 means the user never has to logoff.

usrlog1_kickoff_time specifies the time when the user will be logged off by the system. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970. A value of -1 means the system will   never logoff the user.

 usrlog1_password_age specifies the time in seconds since the user last changed his/her password.

usrlog1_password_can_change specifies the time when the user can change the password. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970. A value of -1 means the user can never change the password.

usrlog1_password_must_change specifies the time when the user must change the password. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970.

usrlog1_computer specifies the computer where the user is logged on.

usrlog1_script_path specifies the relative path to the user logon script.

usrlog1_reserved is reserved with an undefined value.

 The following table defines the valid fields in the user_logon_info_1 structure based upon the return values::

| function return code | usrlog1_code element | Valid elements of   logoff_info_1 |
|---|---|---|
| NERR_Success | NERR_Success | All |
| NERR_Success | NERR_StandaloneLogon | None except usrlog1_code |
| ERROR_ACCESS_DENIED | NERR_PasswordExpired | None except usrlog1_code |
| ERROR_ACCESS_DENIED | NERR_InvalidWorkstation | None except usrlog1_code |
| ERROR_ACCESS_DENIED | NERR_InvalidLogonhours | None except usrlog1_code |
| ERROR_ACCESS_DENIED | NERR_LogonScriptError | None except usrlog1_code |
| ERROR_ACCESS_DENIED | ERROR_ACCESS_DENIED | None except usrlog1_code |
| All other errors | None; the code is meaningless | None |

All of the pointers in this data structure need to be treated specially. The   pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

There is no auxiliary data in the response.

## 10.        NetwkstaUserLogoff

This is a function executed on a remote CIFS server to log on a user. The purpose is to perform some checks and accomplish housekeeping and statistics updates.

The definition is:

```
unsigned short NetWkstaUserLogoff(
    char              *reserved1;
```

```
    char              *reserved2;
    unsigned short    sLevel;
    BYTE              bReqBuffer[54];
    unsigned short    cbReqBuffer;
    REQBUF            pbBuffer;
    REQBUFLEN         cbBuffer;
    unsigned short    *pcbTotalAvail;
);
```

where:

reserved1 and reserved2 are reserved fields and must be null.

sLevel specifies the level of detail returned. The only legal value is 1.

pbReqBuffer points to the request buffer. This buffer contains parameters that need to be sent to the server. The actual value and structure is defined in the Transaction Request Parameters section.

cbReqBuffer specifies the size, in bytes, of the buffer pointed to by the *pbReqBuffer* parameter.   The value must be decimal 54.

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

* The 16 bit function number for NetWkstaUserLogoff which is 133.
* The parameter descriptor string which is "zzWb38WrLh"
* The data descriptor string for the (returned) data which is "WDW"

- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A null pointer
- Another null pointer
- A 16 bit integer with a value of 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- An array of   length 38 bytes. These 38 bytes are defined as

```
char              wlreq1_name[21];           // User Name
char              wlreq1_pad1;         //Pad next field to a word boundary
char              wlreq1_workstation[16];      //ASCII name of computer
```
- A 16 bit integer with a value of decimal 38.
- A 16 bit integer that contains the size of the receive buffer

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|------|-------|-------------|
| NERR_Success | 0 | No errors encountered |
| NERR_StandaloneLogon | 2214 | The logon was not validated by any server |
| NERR_NonValidatedLogon | 2217 | The logon server is running an older software version and cannot validate the logoff |

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or   ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

## Transaction Response Data section

The Transaction response data section contains a data structure user_logoff_info_1 which is defined as:

```
struct user_logoff_info_1 {
        unsigned short  usrlogf1_code;
        unsigned long   usrlogf1_duration;
        unsigned short  usrlogf1_num_logons;
};
```

where:

usrlogf1_code specifies the result and can have the following values:

| Code | Value | Description |
|------|-------|-------------|
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| NERR_InvalidWorkstation | 2240 | The user is not allowed to logon from this computer |

usrlogf1_duration specifies the time in number of seconds for which the user was logged

usrlogf1_num_logons specifies the number of times this user has logged on. A value of -1 indicates the number is unknown.

The following table defines the valid fields in the logoff_info_1 structure based upon the return values::

| function return code | usrlogf11_code element | Valid elements of   logoff_info_1 |
| --- | --- | --- |
| NERR_Success | NERR_Success | All |
| NERR_Success | NERR_StandaloneLogon | None except usrlogf1_code |
| All other errors | None; the code is meaningless | None |

There is no auxiliary data in the response.

# 11.        NetUserGetInfo

This is a function executed on a remote CIFS server to obtain detailed information about a particular user.

The definition is:

```
unsigned short NetUserGetInfo(
   char              *pszUser;
   unsigned short     sLevel;
   RCVBUF            pBuffer;
   RCVBUFLEN         cbBuffer;
   unsigned short    *pcbTotalAvail;
);
```

where:

pszUser points to a null terminated ASCII string signifying the name of the user for which information should be retrieved.

sLevel specifies the level of detail returned. The only legal value is 11.

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

•            The 16 bit function number for NetUserGetInfo which is 56.

- The parameter descriptor string which is "zWrLh"
- The data descriptor string for the (returned) data which is "B21BzzzWDDzzDDWWzWzDWb21W"
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A null terminated ASCII string indicating the user for which information should be retrieved.
- A 16 bit integer with a value of decimal 11 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- A 16 bit integer that contains the size of the receive buffer


## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|---|---|---|
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| ERROR_MORE_DATA | 234 | additional data is available |
| NERR_BufTooSmall | 2123 | The supplied buffer is too small |
| NERR_UserNotFound | 2221 | The user name was not found |

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or  ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

## Transaction Response Data section

The Transaction response data section contains a data structure user_logon_info_1 which is defined as:

```
struct user_info_11 {
        char                usri11_name[21];
        char                usri11_pad;
        char                    *usri11_comment;
        char                *usri11_usr_comment;
        char                *usri11_full_name;
        unsigned short      usri11_priv;
        unsigned long       usri11_auth_flags;
        long                usri11_password_age;
        char                    *usri11_homedir;
        char                *usri11_parms;
        long                usri11_last_logon;
        long                usri11_last_logoff;
        unsigned short      usri11_bad_pw_count;
        unsigned short      usri11_num_logons;
        char                *usri11_logon_server;
        unsigned short      usri11_country_code;
        char                *usri11_workstations;
        unsigned long       usri11_max_storage;
        unsigned short      usri11_units_per_week;
        unsigned char       *usri11_logon_hours;
        unsigned short      usri11_code_page;
    };
```

where:

usri11_name specifies the user name for which information is retrieved

usri11_pad aligns the next data structure element to a word boundary

usri11_comment is a null terminated ASCII comment

usri11_user_comment is a null terminated ASCII comment about the user

usri11_full_name is a null terminated ASCII specifying the full name of the user

usri11_priv specifies the level of the privilege assigned to the user. The possible values are:

| Name | Value | Description |
|------|-------|-------------|
| USER_PRIV_GUEST | 0 | Guest privilege |
| USER_PRIV_USER | 1 | User privilege |
| USER_PRV_ADMIN | 2 | Administrator privilege |

usri11_auth_flags specifies the account operator privileges. The possible values are:

| Name | Value | Description |
| --- | --- | --- |
| AF_OP_PRINT | 0 | Print operator |
| AF_OP_COMM | 1 | Communications operator |
| AF_OP_SERVER | 2 | Server operator |
| AF_OP_ACCOUNTS | 3 | Accounts operator |

usri11_password_age specifies how many seconds have elapsed since the password was last changed.

usri11_home_dir points to a null terminated ASCII string that contains the path name of the user's home directory.

usri11_parms points to a null terminated ASCII string that is set aside for use by applications.

usri11_last_logon specifies the time when the user last logged on. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970.

usri11_last_logoff specifies the time when the user last logged off. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970. A value of 0 means the last logoff   time is unknown.

usri11_bad_pw_count specifies the number of incorrect passwords entered since the last successful logon.

usri11_log1_num_logons specifies the number of times this user has logged on. A value of -1 means the number of logons is unknown.

usri11_logon_server points to a null terminated ASCII string that contains the name of the server to which logon requests are sent. A null string indicates logon requests should be sent to the domain controller.

usri11_country_code specifies the country code for the user's language of choice.

usri11_workstations points to a null terminated ASCII string that contains the names of workstations the user may log on from. There may be up to 8 workstations, with the names separated by commas. A null strings indicates there are no restrictions.

usri11_max_storage specifies the maximum amount of disk space the user can occupy. A value of 0xffffffff indicates there are no restrictions.

usri11_units_per_week specifies the equal number of time units into which a week is divided. This value must be equal to 168.

usri11_logon_hours points to a 21 byte (168 bits) string that specifies the time during which the user can log on. Each bit represents one unique hour in a week. The first bit (bit 0, word 0) is Sunday, 0:00 to 0:59, the second bit (bit 1, word 0) is Sunday, 1:00 to 1:59 and so on. A null pointer indicates there are no restrictions.

usri11_code_page specifies the code page for the user's language of choice

All of the pointers in this data structure need to be treated specially. The   pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

There is no auxiliary data in the response.

# 12. NetWkstaGetInfo

This is a function executed on a remote CIFS server to obtain detailed information about a workstation.

The definition is:

**unsigned short NetWkstaGetInfo**(
    **unsigned short**    *sLevel*;
    **RCVBUF**       pBuffer;
    **RCVBUFLEN**    *cBuffer*;
    **unsigned short**    *\*pcbTotalAvail*;
);

    where:

        sLevel specifies the level of detail returned. The only legal value is 10.

        pbBuffer points to the buffer to receive the returned data.

        cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

        pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetWkstaGetInfo which is 63.
- The parameter descriptor string which is "WrLh"
- The data descriptor string for the (returned) data which is "zzzBBzz".
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A 16 bit integer with a value of decimal 10 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- A 16 bit integer that contains the size of the receive buffer

## Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

## Transaction Response Parameters section

The transaction response parameters section consists of:
- A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
|------|-------|-------------|
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| ERROR_MORE_DATA | 234 | additional data is available |
| NERR_BufTooSmall | 2123 | The supplied buffer is too small |
| NERR_UserNotFound | 2221 | The user name was not found |

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or  ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

## Transaction Response Data section

The Transaction response data section contains a data structure user_logon_info_1 which is defined as:

```
struct user_info_11 {
     char                *wki10_computername;
     char                *wki10_username;
     char                   *wki10_langroup;
     unsigned char     wki10_ver_major;
     unsigned char     wki10_ver_minor;
```

```
        char                      *wki10_logon_domain;
        char                  *wki10_oth_domains;
    };
```

where:

wki10_computername is a pointer to a NULL terminated ASCII string that specifies the name of the workstation.

wki10_username is a pointer to a NULL terminated ASCII string that specifies the user who is logged on at the workstation.

wki10_langroup is a pointer to a NULL terminated ASCII string that specifies the domain to which the workstation belongs.

wki10_ver_major specifies the major version number of the networking software the workstation is running.

wki10_ver_minor specifies the minor version number of the networking software the workstation is running.

wki10_logon domain is a pointer to a NULL terminated ASCII string that specifies the domain for which a user is logged on.

wki10_oth domain is a pointer to a NULL terminated ASCII string that specifies all domains in which the computer is enlisted.

All of the pointers in this data structure need to be treated specially. The   pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

There is no auxiliary data in the response.

# 13.      SamOemChangePassword

This is a function executed on a remote CIFS server to change a user's password.

The definition is:

```
unsigned short SamOemChangePassword(
    uchar              *UserName;
    uchar              *OldPassword;
    uchar              *NewPassword;
);
```

where:

UserName is a pointer to a NULL terminated ASCII string representing the name of the user for which the password should be changed.

OldPassword is a pointer to a NULL terminated ASCII string representing the current password of the user

NewPassword is a pointer to a NULL terminated ASCII string representing the new password of the

## Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for SamOEMChangePassword which is 214.
- The parameter descriptor string which is "zsT"
- The actual parameters as described by the parameter descriptor string.

The parameters are:
- A null terminated ASCII string that represents the name of the user for whom the password is being changed.
- A word with a value of 532 representing the size of the data buffer.

## Transaction Request Data section

The data buffer to be sent consists of 532 bytes of data. The first 516 bytes represent the new password in an encrypted form. The last 16 bytes represent the old password in an encrypted form.

The new password is represented by the structure

```
struct    {
     char NewPasswordBuffer[512];
     long LengthofNewPasswordInBytes;
}
```

The new password is stored in plain text form at the end of the buffer and the length of the new password is stored in the second member of the structure. The whole structure is encrypted using RC4. The RC4 key used is the One Way Transformation (described below) of the old password.

The RC4 encryption of the One Way Transformation of the old password constitutes the last 16 bytes of the data buffer. The RC4 key used is the One Way Transformation of the new password

There is no auxiliary data to send as part of the request.


## One Way Transformation


This section describes the algorithm used by CIFS to apply a one way transformation on data.

Let
E(K, D)
    denote the DES block mode encryption function [5] , which accepts a seven byte key (K) and an eight byte data block (D) and produces an eight byte encrypted data block as its value.

concat(A, B)
    is the result of concatenating A and B

Ex(K,D)
    denote the extension of DES to longer keys and data blocks. If the data to be encrypted is longer than eight bytes, the encryption function is applied to each block of eight bytes in sequence and the results are concatenated together.   If the key is longer than seven bytes, each 8 byte block of data is first completely encrypted using the first seven bytes of the key, then the second seven bytes, etc., appending the results each time.   For example, to encrypt the 16 byte quantity D0D1 with the 14 byte key K0K1,

        Ex(K0K1,D0D1) = concat(E(K0,D0),E(K0,D1),E(K1,D0),E(K1,D1))

head(S, B)
    denote the first B bytes of the byte string S.

swab(S)
    denote the byte string obtained by reversing the order of the bits in each byte of S, i.e., if S is byte string of length one, with the value 0x37 then swab(S) is 0xEC.



The One Way Transformation function is defined as:

```
    OWF = Ex(swab(P14), N8)
```

Where
- P14 is the data to encrypted. If P14 is the user's password, it is a clear, upper-cased text string, padded with blanks
- N8 is an 8 byte string whose value is available from Microsoft upon request

## Transaction Response Parameters section

The transaction response parameters section consists of:
-        A 16 bit word indicating the return status. The possible values are:

| Code | Value | Description |
| --- | --- | --- |
| NERR_Success | 0 | No errors encountered |
| ERROR_ACCESS_DENIED | 5 | User has insufficient privilege |
| ERROR-INVALID-PASSWORD | 86 | The specified password is invalid |
| NERR_PasswordCantChange | 2243 | The password cannot be changed |
| NERR_PasswordTooShort | 2246 | The password is too short |

## Transaction Response Data section

There is no Transaction Response Data to receive

There is no auxiliary data in the response.


## 14.        Author's Addresses

Paul Leach
Dilip Naik
Microsoft
1 Microsoft Way
Redmond, WA   98052
paulle@microsoft.com
v-dilipn@microsoft.com


## 15.        Appendix A

### Transaction SMBs

These SMBs are used both to retrieve bulk data from the server (e.g.: enumerate shares, etc.) and to change the server's state (EG: add a new share, change file permissions, etc.)   Transaction requests are also unusual because they can have a multiple part request and/or a multiple part response.   For this reason, transactions are handled as a set of sequenced commands to the server. Each part of a request is sent as a sequenced command using the same *Mid* value and an increasing *Seq* value.   The server responds to each request piece except the last one with a response indicating that the server is ready for the next piece.   The last piece is responded to with the first piece of the result data.   The client then sends a transaction secondary SMB with *ParameterDisplacement* set to the number of parameter bytes received so far and *DataDisplacement* set to the number of data bytes received so far and *ParameterCount*, *ParameterOffset*, *DataCount*, and *DataOffset* set to zero (0).   The server responds with the next piece of the transaction result.   The process is repeated until all of the response information has been received. When the transaction has been completed, the redirector must send another sequenced command (an echo SMB will do fine) to the server to allow the server to know that the final piece was received and that resources allocated to the transaction command may be released.

The flow is as follows, where (S) is the *SequenceNumber*,   (N) is the number of request packets to be sent from the client to the server, and (M) is the number of response packets to be sent by the server to the client:

| Client | <-> | Server |
|========================|====|==========================|
| SMB(S)   Transact | -> | |
|  | <- | OK (S) send more data |
| [ repeat N-1 times: |  | |
|     SMB(S+1) Transact secondary | -> | |
|  | <- | OK (S+1) send more data |
|     SMB(S+N-1) |  | |
| ] |  | |
|  | <- | OK (S+N-1) transaction response (1) |
| [ repeat M-1 times: |  | |
|     SMB(S+N) Transact secondary | -> | |
|  | <- | OK (S+N) transaction response (2) |
|     SMB(S+N+M-2) Transact secondary | -> | |
|  | <- | OK (S+N+M-2] transaction response (M) |
|  ] |  | |
| SMB(S+N+M-1) Echo | -> | |
|  | <- | OK (S+N+M-1) echoed |

In order to allow the server to detect clients which have been powered off, have crashed, etc., the client must send commands to the server periodically if it has resources open on the server.   If nothing has been received from a client for awhile, the server will assume that the client is no longer running and disconnect the client.   This includes closing any files that the client had open at the time and releasing any resources being used on behalf of the client.   Clients should at least send an echo SMB to the server every four (4) minutes if there is nothing else to send.   The server will disconnect clients after a configurable amount of time which cannot be less than five (5) minutes.   (Note: the NT server has a default timevalue of 15 minutes.)

### 15.1.1        TRANSACTIONS

`SMB_COM_TRANSACTION` performs a symbolically named transaction.   This transaction is known only by a name (no file handle used).   `SMB_COM_TRANSACTION2` likewise performs a transaction, but a word parameter is used to identify the transaction instead of a name.   `SMB_COM_NT_TRANSACTION` is used for commands that potentially need to transfer a large amount of data (greater than 64K bytes).

### *15.1.1.1        SMB_COM_TRANSACTION AND SMB_COM_TRANSACTION2 FORMATS*

| Primary Client Request ============================= | Description ================================== |
|---|---|
| Command | SMB_COM_TRANSACTION or SMB_COM_TRANSACTION2 |
| UCHAR WordCount; | Count of parameter words;    value = (14 + SetupCount) |
| USHORT TotalParameterCount; | Total parameter bytes being sent |
| USHORT TotalDataCount; | Total data bytes being sent |
| USHORT MaxParameterCount; | Max parameter bytes to return |
| USHORT MaxDataCount; | Max data bytes to return |
| UCHAR MaxSetupCount; | Max setup words to return |
| UCHAR Reserved; | |
| USHORT Flags; | Additional information: |
| | bit 0 - also disconnect TID in *TID* |
| | bit 1 - one-way transaction (no resp) |
| ULONG Timeout; | |
| USHORT Reserved2; | |
| USHORT ParameterCount; | Parameter bytes sent this buffer |
| USHORT ParameterOffset; | Offset (from header start) to Parameters |
| USHORT DataCount; | Data bytes sent this buffer |
| USHORT DataOffset; | Offset (from header start) to data |
| UCHAR SetupCount; | Count of setup words |
| UCHAR Reserved3; | Reserved (pad above to word) |
| USHORT Setup[SetupCount]; | Setup words (# = SetupWordCount) |
| USHORT ByteCount; | Count of data bytes |
| STRING Name[]; | Name of transaction (NULL if SMB_COM_TRANSACTION2) |
| UCHAR Pad[]; | Pad to SHORT or LONG |
| UCHAR Parameters[ParameterCount]; | Parameter bytes (# = ParameterCount) |
| UCHAR Pad1[]; | Pad to SHORT or LONG |
| UCHAR Data[ DataCount ]; | Data bytes (# = DataCount) |

| Interim Server Response ============================= | Description ================================== |
|---|---|
| UCHAR WordCount; | Count of parameter words = 0 |
| USHORT ByteCount; | Count of data bytes = 0 |

| Secondary Client Request | Description |
|---|---|
| ============================ | ================================ |
| Command | SMB_COM_TRANSACTION_SECONDARY |
| | |
| UCHAR WordCount; | Count of parameter words = 8 |
| USHORT TotalParameterCount; | Total parameter bytes being sent |
| USHORT TotalDataCount; | Total data bytes being sent |
| USHORT ParameterCount; | Parameter bytes sent this buffer |
| USHORT ParameterOffset; | Offset (from header start) to Parameters |
| USHORT ParameterDisplacement; | Displacement of these Parameter bytes |
| USHORT DataCount; | Data bytes sent this buffer |
| USHORT DataOffset; | Offset (from header start) to data |
| USHORT DataDisplacement; | Displacement of these data bytes |
| USHORT Fid; | *FID* for handle based requests, else 0xFFFF.  This field is present only if this is an SMB_COM_TRANSACTION2 request. |
| USHORT ByteCount; | Count of data bytes |
| UCHAR Pad[]; | Pad to SHORT or LONG |
| UCHAR Parameters[ParameterCount]; | Parameter bytes (# = ParameterCount) |
| UCHAR Pad1[]; | Pad to SHORT or LONG |
| UCHAR Data[DataCount]; | Data bytes (# = DataCount) |

| Server Response | Description |
|---|---|
| ============================ | ================================ |
| UCHAR WordCount; | Count of data bytes; value = 10 + *SETUPCOUNT* |
| USHORT TotalParameterCount; | Total parameter bytes being sent |
| USHORT TotalDataCount; | Total data bytes being sent |
| USHORT Reserved; | |
| USHORT ParameterCount; | Parameter bytes sent this buffer |
| USHORT ParameterOffset; | Offset (from header start) to Parameters |
| USHORT ParameterDisplacement; | Displacement of these Parameter bytes |
| USHORT DataCount; | Data bytes sent this buffer |
| USHORT DataOffset; | Offset (from header start) to data |
| USHORT DataDisplacement; | Displacement of these data bytes |
| UCHAR SetupCount; | Count of setup words |
| UCHAR Reserved2; | Reserved (pad above to word) |
| USHORT Setup[SetupWordCount]; | Setup words (# = SetupWordCount) |
| USHORT ByteCount; | Count of data bytes |
| UCHAR Pad[]; | Pad to SHORT or LONG |
| UCHAR Parameters[ParameterCount]; | Parameter bytes (# = ParameterCount) |
| UCHAR Pad1[]; | Pad to SHORT or LONG |
| UCHAR Data[DataCount]; | Data bytes (# = DataCount) |

# 16.     Appendix B

## 16.1        *Marshaling and unmarshaling using descriptor strings*

TBD. This will be a note to explain how the descriptor strings can be used to drive a marshaling engine that can automatically marshal and unmarshal RAP messages and call local APIs whose calling sequences closely match the format of the RAP services.