Network Working Group
INTERNET-DRAFT
draft-leach-cifs-print-spec-00.txt
Category: Informational
Expires June 31, 1997

Paul J. Leach, Microsoft
Dilip C. Naik, Microsoft

January 31, 1997

CIFS Printing Specification

Preliminary Draft

STATUS OF THIS MEMO

THIS IS A PRELIMINARY DRAFT OF AN INTERNET-DRAFT. IT DOES NOT REPRESENT THE CONSENSUS OF ANY WORKING GROUP.

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited. Please send comments to the authors or the CIFS mailing list at <cifs@listserv.msn.com>. Discussions of the mailing list are archived at <URL:http://microsoft.ease.lsoft.com/archives/cifs.html>.

ABSTRACT

This specification defines how clients may submit print requests to a server using SMBs . The specification also details how clients may administer printing of the print requests they create, using SMBs defined in the Common Internet File System specification.

Table of Contents

1. Objective

This document describes

- how CIFS clients accomplish printing on CIFS servers acting as print servers.
- how CIFS clients administer printing on CIFS servers.

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) in general and the CIFS Remote Administration Protocol in particular.

3. Printing overview

A CIFS client opens a print file on a CIFS server in a manner similar to opening an ordinary file, but using a different SMB (described in section 4.1). The CIFS client then writes to the file. When the CIFS client closes the file, the CIFS print server considers the file to be a print job or print request that needs to be printed.

The CIFS print server implements a concept of print queues. A print queue is simply an ordered set of print requests or print jobs. Every print job is associated with a print queue. A client can control which print queue a print job is associated with. A printer queue stores print jobs and sends them one by one to a printer. A print queue may have multiple physical printers associated with it. Different queues may share the same printer. Print queues allow administrative convenience such as selection of a printer, priority assignment for printing, controlling times during which jobs may print, etc. .

Section 4 describes how a client may generate a print job.

Section 6 describes how a client may enumerate print queues and retrieve information about a particular print queue.

Section 7 describes how a client may manipulate print jobs, causing print jobs to be paused, resumed or deleted.

Print queues and print jobs are manipulated using the CIFS Remote Administration protocol. The CIFS specification includes details on Remote Administration protocol. For convenience, details have been duplicated (from the CIFS specification) into Appendix A and Appendix B. Note that the CIFS specification should be considered a more authoriative source of information, as compared to Appendix A and B.

4. Creating a Print Job

A CIFS client creates a print job (or a print request) by opening a print file, writing to the print file and then closing it. A print file differs from an ordinary file in that a CIFS server tracks a print file and deletes it automatically when the printing is complete.

A print job is associcated with a print queue. Different print queues may have different characteristics and may print on different printers. A client can control which print queue a print job (created by the client) is associated with. The CIFS print server shares different queues. A client first creates a connection via a SessionSetupAndX followed by a TreConnectAndX SMB, specifying the appropriate print queue share in the TreeConnectAndX SMB. Refer to the CIFS document for details on the TreeConnectAndX SMB. The TreeConnectAndX will return a Tree Id (Tid) if the SMB succeeds. This Tid is used when opening a print file.

4.1 OPEN_PRINT_FILE: Create Print Spool file

This message is sent to create a new printer file which will be deleted once it has been closed and printed. Complete understanding of this message requires familiarity with the CIFS specification.

Client Request ================================	Description ================================
UCHAR WordCount; USHORT SetupLength; USHORT Mode; USHORT ByteCount; UCHAR BufferFormat; STRING IdentifierString[];	Count of parameter words = 2 Length of printer setup data 0 = Text mode (DOS expands TABs) 1 = Graphics mode Count of data bytes; min = 2 0x04 Identifier string

TID in the SMB header must refer to a printer resource type.

SETUPLENGTH is the number of bytes in the first part of the resulting print spool file which contains printer-specific control strings.

MODE can have the following values:

0	Text mode. The server may optionally expand tabs to a series of spaces.
1	Graphics mode. No conversion of data should be done by the server.

IDENTIFIERSTRING can be used by the server to provide some sort of per-client identifying component to the print file.

Server Response	Description
UCHAR WordCount; USHORT Fid; USHORT ByteCount;	Count of parameter words = 1 File handle Count of data bytes = 0

FID is the returned handle which may be used by subsequent write and close operations. When the file is finally closed, it will be sent to the spooler and printed.

4.2 CLOSE_PRINT_FILE: Close and Spool Print Job

This message invalidates the specified file handle and queues the file for printing. Complete understanding of this message requires familiarity with the CIFS specification.

Client Request	Description
UCHAR WordCount; USHORT Fid; USHORT ByteCount;	Count of parameter words = 1 File handle Count of data bytes = 0

FID refers to a file previously created with SMB_COM_OPEN_PRINT_FILE. On successful completion of this request, the file is queued for printing by the server.

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

Servers which negotiate dialects of LANMAN1.0 and newer allow all the other types of *FID* closing requests to invalidate the *FID* and begin spooling.

5. Remote Administration Protocol and Documentation Conventions

Print queue and print job related management functions are accomplished using the CIFS Remote Administration Protocol (RAP). Complete details may be found in the CIFS specification and in Appendix A. Persons unfamiliar with the RAP specification are strongly advised to read the CIFS specification or at least Appendix A at this stage. Sections that follow describe how a CIFS client queries information about print queues and print jobs and administers print jobs These descriptions assume knowledge of the CIFS RAP specification.

6. Print Queues and related functions

A CIFS server can enumerate all print queues on a given server using the DosPrintQEnum function. Once the CIFS client knows the names of each print queue, the CIFS client can then obtain information about each print queue using the DosPrintQGetInfo function. Both of these functions are executed on the remote print server using the CIFS Remote Administration Protocol, fully detailed in the CIFS document as well as Appendix A and Appendix B.

6.1 Data structures related to Print queues

This section describes data structures used to describe print queues. Data structures corresponding to print queue levels 0 , 1 and 2 are obsolete.

6.1.1 Printer Queue Information (Level 3)

The PRQINFO_3 data structure describes a particular printer queue. The DosPrintQEnum and DosPrintQGetInfo functions (described below) return data in this format when the desired level of information is set to level 3.

```
struct    PRQINFO_3 {
      char                *pszName;
      unsigned short  Priority;
      unsigned short  Starttime;
      unsigned short  UntilTime;
      unsigned short  Pad1;
      char                *pszSepFile;
      char                *pszPrProc;
      char                *pszParms;
      char                *pszComment;
      unsigned short  Status;
      AUXCOUNT        cJobs;
      char                *pszPrinters;
      char                *pszDriverName
      void                *pDriverData;
}
```

where:

pszName points to a null terminated ASCII string that contains the queue name.

Priority contains an unsigned short integer specifying the printer queue priority. The value can range from 1(highest) to 9 (lowest). When two printer queues print to the same printer, the print jobs from the one with the higher priority print first.

Untiltime contains an unsigned short integer specifying the time of day a printer queue becomes inactive and stops sending print jobs to printers. This value represents the number of minutes since midnight (00:00).

Starttime contains an unsigned short integer specifying the time of day a printer queue can start sending print jobs to printers. This value represents the number of minutes since midnight (00:00).

pszSepFile points to a null terminated ASCII string that represents the pathname to a seperator page file. The seperator page contains formatting information about the pages that separate print jobs.

pszPrProc points to a null terminated ASCII string that represents the name of the print preprocessor. A null pointer or null string inidcates the default print preprocessor.

pszDestinations points to a null terminated ASCII string that contains a list of print destinations for the print queue. This is a multi-valued property and the values are separated by spaces.

pszParms points to a null terminated ASCII string that contains parameters required by printer queues.

pszComment points to a null terminated ASCII string that contains a comment about the print queue.

Status contains an unsigned short integer that specifies the status of a printer queue. Possible values are:

Code	Value	Description
PRQ_ACTIVE	0	Active
PRQ_PAUSE	1	Paused
PRQ_ERROR	2	Error Occurred
PRQ_PENDING	3	Deletion pending

cJobs contains an unsigned short integer representing the number of print jobs currently in the print queue.

pszDriverName points to a null terminated ASCII string representing the default device driver for the queue. If this field is null, the pDriverData field is not used.

pDriverData points to the device driver data for the default driver.

6.1.2 Printer Queue Information (Level 4)

At this level, the returned information consists of a level 3 print queue data structure (as described in section 5.1.1) followed by a PRJINFO_2 data structure for each print job in the queue. The PRJINFO_2 data structure is described in section 7.1.

6.1.3 Printer Queue Information (Level 5)

At level 5, the returned data structure is defined as:

```
struct    PRQINFO_5 {
     char    *pszName;
}
```

where:

pszName points to a null terminated ASCII string that contains the queue name.

6.2 DosPrintQEnum

The DosPrintQEnum function lists all printer queues on a server. The definition is:

```
unsigned short DosPrintQEnum(
     unsigned short     sLevel;
     RCVBUF             pbBuffer;
     RCVBUFLEN          cbBuffer;
     ENTCOUNT           pcReturned;
     unsigned short     *pcTotalAvail;
);
```

where:

sLevel specifies the level of detail returned. Legal values are 3, 4 and 5 .

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.

pcReturned points to a 16 bit variable that receives a count of the total number of entries (queues) returned. This count is valid only if **DosPrintQEnum** 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 entries (queues) available. This count is valid only if **DosPrintQEnum** returns the NERR_Success or ERROR_MORE_DATA values.

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintQEnum which is 69.
- The parameter descriptor string which is "WrLeh"
- The data descriptor string for the returned data is "zWWWWzzzzWWzzl" if sLevel (the level of desired information) is 3. This corresponds to the data structure PRQINFO_3There is no auxiliary data in the response when the level of desired information is 3.
- The data descriptor string for the returned data is "zWWWWzzzzWNzzl" if sLevel (the level of desired information) is 4. This corresponds to the data structure PRQINFO_3. The descriptor for the auxiliary data returned is "WWzWWDDzz" when the level of desired information is 4. This corresponds to the PRJINFO_2 data structure.
- The data descriptor string for the returned data is "z" if sLevel (the level of desired information) is 5. The "z" indicates a null terminated ASCII string representing the name of the print queue. . There is no auxiliary data descriptor for this level of information.
- The actual parameters as described by the parameter descriptor string. These are:
- A 16 bit integer with a value of 3, 4 or 5 (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	The user does not have required priveleges
ERROR_MORE_DATA	234	Additional data is available
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server
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 Transaction response data section consists of a series of data structures. The number of the data structures is equal to the number or entries being returned, which is the third value in the Transaction response parameter section.

At information level 3, a series of PRQINFO_3 data structures are returned. There is no auxiliary data at this information level.

At information level 4, a series of PRQINFO_3 data structures are returned. There is also auxiliary data present in this case. For each print job in each print queue, a PRJINFO_2 structure is returned in the auxiliary data. For each print queue, the data descriptor string element “N” denotes the number of PRJINFO_2 data structures present in the auxiliary data for that queue. This also denotes the number of print jobs associated with that print queue.

At information level 5, a PRQINFO_5 data structure is returned for each print queue. There is no auxiliary data in the response when the level of desired information is 5.

As per the RAP specification, all pointers in any of the data structures returned need to be treated specially in the prescribed manner.

6.3 DosPrintQGetInfo

The DosPrintQGetInfo function retrieves information about a particular print queue on a CIFS server. The definition is:

```
unsigned short DosPrintQGetInfo(
     char               *pszQueueName;
     short              sLevel;
     RCVBUF             pbBuffer;
     RCVBUFLEN          cbBuffer;
     unsigned short     *pcbTotalAvail;
);
```

where:

pszQueueName points to an ASCII null-terminated string specifying the name of the queue for which information should be retrieved.

sLevel specifies the level of detail returned. (Legal values are 3, 4, and 5)

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 **DosPrintQGetInfo** returns the NERR_Success or ERROR_MORE_DATA values.

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintQGetInfo which is 70.
- The parameter descriptor string which is “zWrLh”
- The data descriptor string for the returned data is “zWWWWzzzzWWzzl” if sLevel (the level of desired information) is 3. This corresponds to the data structure PRQINFO_3. There is no auxiliary data in the response when the level of desired information is 3.

- The data descriptor string for the returned data is “zWWWWzzzzWNzzl” if sLevel (the level of desired information) is 4. This corresponds to the data structure PRQINFO_3. The descriptor for the auxiliary data returned is “WWzWWDDzz” when the level of desired information is 4. This corresponds to the PRJINFO_2 data structure.
- The data descriptor string for the returned data is “z” if sLevel (the level of desired information) is 5. The “z” indicates a null terminated ASCII string representing the name of the print queue. . There is no auxiliary data descriptor for this level of information.
- The actual parameters as described by the parameter descriptor string. These are:
    - A null terminated ASCII string denoting the name of the print queue for which information should be retrieved.
    - A 16 bit integer with a value of 3, 4 or 5 . 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_QNotFound	2150	The specified queue name is invalid
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server

- A 16 bit “converter” word. The value is up to the server to decide.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success, ERROR_MORE_DATA or NERR_BufTooSmall

Transaction Response Data Section

The Transaction response data section consists of a single data structure

At information level 3, a PRQINFO_3 data structure is returned. There is no auxiliary data at this information level.

At information level 4, a PRQINFO_3 data structure is returned. This data structure describes the print queue of interest. There is also auxiliary data present in this case. For each print job in the print queue, a PRJINFO_2 structure is returned in the auxiliary data. The data descriptor string element “N” denotes the number of PRJINFO_2 data structures present in the auxiliary data for that queue. This also denotes the number of print jobs associated with that print queue.

At information level 5, a PRQINFO_5 data structure is returned for each print queue. There is no auxiliary data in the response when the level of desired information is 5.

As per the RAP specification, all pointers in any of the data structures returned need to be treated specially in the prescribed manner.

7. Print Jobs and manipulating Print Jobs

Once a CIFS client has located a print queue, the client can then enumerate jobs within that queue using the DosPrintJobEnum function. A CIFS client may also obtain print job information by means of the DosPrintQEnum and DosPrintQGetInfo services, specifying a desired information level of 4. Once the CIFS client has a list of job identifiers, it can obtain detailed information about any print job using the DosPrintJobGetInfo function. For print jobs initiated by the client, and which are not yet printing, the CIFS client can pause, resume or delete the print jobs using DosPrintJobPause, DosPrintJobContinue and DosPrintJobDel functions respectively. All of these DosPrintJobX services are executed on the remote print server using the CIFS Remote Administration Protocol described in the CIFS document as well as in Appendix A and Appendix B.

7.1 Data Structures related to Print Jobs

```
struct PRJINFO_0 {
     unsigned short JobId
}
```

where:

JobId is a 16 bit integer that uniquely specifies a print job within a printer queue. The JobID is unique on a server. A combination of the server name and JobId is sufficient to uniquely identify a particular print job.

```
struct PRJINFO_2 {
     unsigned short     JobId;
     unsigned short     Priority;
     char               *pszUserName;
     unsigned short     Position;
     unsigned short     Status;
     unsigned long      Submitted;
     unsigned long      Size;
     char               *pszComment;
     char               *pszDocument;
}
```

where:

JobId is a 16 bit integer that uniquely specifies a print job within a printer queue. The JobID is unique on a server. A combination of the server name and JobId is sufficient to uniquely identify a particular print job.

Priority is a 16 bit integer that specifies the print job priority. This varies from a value of 1 (lowest priority) to 99 (highest priority. Higher priority jobs print first. When 2 jobs have the same priority, the older job prints first.

pszUserName is a pointer to a null terminated ASCII string that specifies the name of the user who submitted the print job.

Position specifies the position of the print job within the print queue. If the value is 1, this print job prints next.

Status is an integer used as a status flag. The values and meanings of the various bits are:

Bits	Code	Value	Description
0-1	PRJ_QS_QUEUED	0	Print job is queued
0-1	PRJ_QS_PAUSED	1	Print job is paused
0-1	PRJ_QS_SPOOLING	2	Print job is spooling
0-1	PRJ_QS_PRINTING	3	Print job is printing, bits 2-11 are valid

Bit	Code	Value	Description

2	PRJ_COMPLETE	0x0004	Print job is complete
3	PRJ_INTERV	0x0008	an error occurred, pszStatus may contain a comment explaining the error
4	PRJ_ERROR	0x0010	Print job is spooling
5	PRJ_DESTOFFLINE	0x0020	The print destination is offline
6	PRJ_DESTPAUSED	0x0040	The print destination is paused
7	PRJ_NOTIFY	0x0080	An alert is raised
8	PRJ_DESTNOPAPER	0x0100	The print destination is out of paper
9	PRJ_DESTFORMCHG	0x0200	The printer is waiting for a form change
10	PRJ_DESTCRTCHG	0x0400	The printer is waiting for a cartridge change
11	PRJ_DESTENCHG	0x0800	The printer is waiting for a pen change
15	PRJ_ PRINTING	0x8000	An alert indicates the job was deleted

pszStatus points to an ASCII string that contains a comment about the status of the job. This element contains valid data only when the job is printing and an error occurs. This element may be null or point to a null string.

Submitted contains an unsigned long integer specifying when the user submitted the job. This is stored as the number of seconds elapsed since 00:00:00 Jan 1st, 1970.

Size contains an unsigned long integer that specifies the size of the print job in terms of number of bytes.

pszComment points to a null terminated ASCII string that contains a comment about the print job.

pszDocument points to a null terminated ASCII string that contains the name of the document.

7.2 DosPrintJobEnum

The DosPrintJobEnum service lists print jobs in the specified printer queue. The definition is:

```
unsigned short DosPrintJobEnum(
     char               *pszQueueName;
     short              sLevel;
     RCVBUF             pbBuffer;
     RCVBUFLEN          cbBuffer;
     unsigned short     *pcbTotalAvail;
);
```

where:

pszQueuename points to a null-terminated string specifying the name of the print queue for which print jobs should be enumerated.

sLevel specifies the level of detail returned. (Legal values are 0 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 the return value is NERR_Success or ERROR_MORE_DATA .

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintJobEnum which is 76.
- The parameter descriptor string which is “zWrLeh”
- The data descriptor string for the returned data is “z” if sLevel (the level of desired information) is 0. This corresponds to the data structure PRJINFO_0 already described.
- The data descriptor string for the returned data is “WWzWWDDzz” if sLevel (the level of desired information) is 2. This corresponds to the PRJINFO_2 data structure.already described.
- The actual parameters as described by the parameter descriptor string. These are:
    - A null terminated ASCII string denoting the name of the print queue which contains the print job of interest.
    - A 16 bit integer with a value of 0 or 2 . This represents the level of detail the server is expected to return
    - A 16 bit integer that contains the size of the receive buffer.

There is no data or auxiliary data that is sent as part of the Transaction request.

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_QNotFound	2150	The specified queue name is invalid
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server

- A 16 bit “converter” word.
- A 16 bit number representing the total number of available entries. This has meaning only if the return status is NERR_Success, ERROR_MORE_DATA or NERR_BufTooSmall

Transaction Response Data Section

The Transaction response data section consists of a data structures.

At information level 0, a series of PRJINFO_0 data structures are returned. The number of structure is equal to the value in the third parameter in the response parameter section. There is no auxiliary data at this information level.

At information level 2, a series of PRJINFO_2 data structures are returned. The number of such data structures returned is equal to the value in the third parameter of the response parameter section. There is no auxiliary data present in this case.

As per the RAP specification, all pointers in any of the data structures returned need to be treated specially in the prescribed manner.

There is no auxiliary data in the response.

7.3 DosPrintJobGetInfo

The DosPrintJobGetInfo service retrieves information about a particular print job. The definition is:

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

where:

JobId specifies identity of the print job for which information should be retrieved.

Level specifies the level of detail returned. (Legal values are 0 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 **DosPrintJobGetInfo** returns the NERR_Success or ERROR_MORE_DATA values.

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintJobGetInfo which is 77.
- The parameter descriptor string which is “WWrLh”
- The data descriptor string for the returned data is “z” if sLevel (the level of desired information) is 0. This corresponds to the data structure PRJINFO_0 already described.
- The data descriptor string for the returned data is “WWzWWDDzz” if sLevel (the level of desired information) is 2. This corresponds to the PRJINFO_2 data structure.already described.
- The actual parameters as described by the parameter descriptor string. These are:
    - A 16 bit integer specifying the identity of the job for which information should be retrieved.
    - A 16 bit integer with a value of 0 or 2 . This represents the level of detail the server is expected to return
    - A 16 bit integer that contains the size of the receive buffer.

There is no data or auxiliary data that is sent as part of the Transaction request.

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	The user does not have required priveleges
ERROR_MORE_DATA	234	Additional data is available
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server
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, ERROR_MORE_DATA or NERR_BufTooSmall

Transaction Response Data Section

The return data section consists of a PRJINFO_0 data structure if the desired level of information is 0. The return data section consists of a PRJINFO_2 data structure if the desired level of information is 2. These have already been detailed.

Note that the pointers in the data structure in data structure PRJINFO_2 need to be treated specially. The high 16 bit word needs to be ignored. The converter word returned in the response parameters section needs to be subtracted from the low 16 bit value to locate the actual offset of the item within the response buffer sent by the server.

There is no auxiliary data to receive.

DosPrintJobPause

DosPrintJobPause pauses a print job in a printer queue. The definition is:

```
unsigned short DosPrintJobPause(
     unsigned short JobId;
);
```

where:

JobId specifies the identity of the print job that should be paused

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintJobPause which is 82.
- The parameter descriptor string which is “W”
- The data descriptor string is null.
- The actual parameters as described by the parameter descriptor string. This consists of just a 16 bit integer representing the JobId, identifying the job to be paused.

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
NERR_BadTransactConfig	2141	The server is not configured for transactions, IPC$ is not shared
NERR_JobNotFound	2151	The specified print job could not be located
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server
NERR_JobInvalidState	2164	The operation cannot be performed on the job in it's current state(job is already printing)

Transaction Response Data Section

There is no data or auxiliary data in the response.

7.4 DosPrintJobContinue

DosPrintJobContinue allows a paused print job to resume printing. The definition is:

```
unsigned short DosPrintJobContinue(
     unsigned short JobId;
);
```

where:

JobId specifies the identity of the print job that should resume printing

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintJobContinue which is 83.
- The parameter descriptor string which is "W"
- The data descriptor string is null.
- The actual parameters as described by the parameter descriptor string. This consists of just a 16 bit integer representing the JobId (identifies job to be paused)

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

NERR_BadTransactConfig	2141	The server is not configured for transactions, IPC$ is not shared
NERR_JobNotFound	2151	The specified print job could not be located
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server
NERR_JobInvalidStatus	2164	The operation cannot be performed on the print job in it's current state

Transaction Response Data Section

There is no data or auxiliary data in the response.

7.5 DosPrintJobDel

DosPrintJobDel deletes a print job from a printer queue. The definition is:

```
unsigned short DosPrintJobDel(
     unsigned short JobId;
);
```

where:

JobId specifies the identity of the print job that should be deleted

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The function number for DosPrintJobDel which is 81.
- The parameter descriptor string which is "W"
- The data descriptor string is null.
- The actual parameters as described by the parameter descriptor string. This consists of just a 16 bit integer representing the JobId , identifying the job to be paused.

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
NERR_BadTransactConfig	2141	The server is not configured for transactions, IPC$ is not shared
NERR_JobNotFound	2151	The specified print job could not be located
NERR_ProcNotRespond	2160	The print process is not responding
NERR_SpoolerNotLoaded	2161	The spooler is not started on the remote server

Transaction Response Data Section

There is no data or auxiliary data in the response.

8. Author's Addresses

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

9. 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 Transact2 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 Transact2 SMB is always set to "\PIPE\LANMAN". The Transact2 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 Transact2 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 Transact2 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*.

10. Appendix A - Remote Administration Protocol

A RAP service request is sent to the server encapsulated in a Transact2 request SMB and the server sends back a Transact2 SMB response. An attribute of the Transact2 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 Transact2 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 Transact2 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.

10.1 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.

10.2 Descriptors

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

10.2.1 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

10.2.2 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

10.2.3 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.

10.3 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 Transact2 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.

10.4 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.

10.5 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.

10.6 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.

11. Appendix B

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 timeout value of 15 minutes.)

11.1.1 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).

11.1.1.1 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)