SEPTEMBER 1988

WRL
Technical Note TN-4

TCP/IP PrintServer: Print Server Protocol

Brian K. Reid
Christopher A. Kent

Western Research Laboratory 100 Hamilton Avenue Palo Alto, California 94301 USA

TCP/IP PrintServer
Print Server Protocol

Brian K. Reid
Christopher A. Kent

September, 1988

Abstract

The TCP/IP PrintServer is a printer that uses Internet protocols to communicate with its clients. The protocol is called the Print Server Protocol. It is a record-oriented protocol layered on top of TCP.

This document is the specification of the Print Server Protocol. It includes both the client protocols (those used to print files on the printer) and the management protocols (those used to control access and accounting and configuration of the printer).

digital **Western Research Laboratory** 100 Hamilton Avenue Palo Alto, California 94301 USA

1. Introduction

In the era of line printers, the way that one communicated with a printer was to send it data, which it printed. No protocol was needed. Modern high-speed PostScript printers are connected via an Ethernet. To print a file on such a print server, one must cope with network connections, access control, error recovery, remote accounting, and support for various features of the PostScript language such as file I/O.

The PrintServer protocol is an Internet protocol, based on the Internet architecture and IP packets, that is built on top of TCP. TCP provides full-duplex reliable, flow-controlled character stream communication between two nodes on a network. The PrintServer protocol is a record-oriented protocol that uses TCP streams as its transport mechanism. The PrintServer protocol is used to communicate with, and control, an Internet-connected PostScript PrintServer.

This document describes the PrintServer protocol. Examples are taken from the Unix operating system because it is so widely known, but the PrintServer protocol is in no way limited to use with Unix and Unix-like systems. Any computer able to implement TCP/IP can use the PrintServer protocol.

2. Concepts

A *PrintServer* is a computer connected to a computer network. It is a PostScript printer, but it has its own Internet address, and it uses the Internet protocols to communicate with its clients. A *client* is a program on some other computer that wishes to communicate with the PrintServer. There are two kinds of clients: *ordinary clients* and *management clients*. An ordinary client is a program that wishes to print a file or enquire about the status of the printer. A management client is a privileged program that provides system services to the printer and that receives and records accounting information. Different management clients offer different services, and the details are not part of the Print Server Protocol, but rather are part of the private agreement between the server and its management client.

Printing is done in units called *print sessions*, or simply *sessions*. A session is an uninterrupted connection from a client to a PrintServer. A client opens a session, then for each file to be printed, it opens a job within that session, prints the file, and closes the job. At the end of the session, the client releases the network connection, and another client is free to print.

Any node in the Internet can communicate with any other node in the Internet. Owners of PostScript PrintServers may wish to restrict or control access to them. Owners may also wish to keep an accounting of printer usage. The management client provides these and other services.

3. Protocols

Communication with the print server takes place using a record-oriented protocol that is layered on top of TCP. Since TCP provides a reliable stream, there is no additional data integrity or sequence checking in the print server protocols.

3.1. Record and data formats

All records are freeform strings of ANSI characters. The beginning of each record is marked by a 1-byte synchronization character (002, control-B) ; the end of the record is determined by the *Length* field of the record, as described by the following syntax:

```
<Line>        ::= <sync byte> <Opcode> <Id> <Length><1space>
                      <Data> <ignored>
<sync byte>   ::= (002)
<Opcode>      ::= [A-Z]([A-Z,0-9])*
<Id>          ::= <Integer>
<Length>      ::= <Integer> 0 <= Length <= 1024
<1space>      ::= exactly one space character
<arbitrary>   ::= <any character 000 to 0255>
<Data>        ::= (<arbitrary>)*
<ignored>     ::= (<arbitrary>)*
```

The *Data* can contain imbedded newline characters or any other character.

Except for the *Data* field, which is preceded by exactly one space, fields are delimited by one or more space characters (octal 40). Other characters that resemble spaces, such as tabs, are not treated as delimiters. The *Data* field can contain space characters; it is bounded by the *Length* field and not by a delimiter character. *Opcode* field indicates what kind of data or command the record is. The *Id* field is an optional record identification or sequence number that can be used, if needed, to differentiate replies from multiple outstanding requests. The *Length* field is the number of bytes in the array *Data*. There is exactly one byte of white space after *Length* and before *Data*. This is to permit leading blanks in the *Data* field. The last character of *Data* is determined by the value of *Length*. All characters that come after the last character of *Data* but before the next *sync byte* are ignored, but an implementation is free to consider this situation to be a bug warranting a nonfatal warning message.

When the *Data* field is used to pass more than one string argument, the multiple arguments are combined in a format called a "list of values". It is much like a Unix environment vector, or, alternatively, much like the "keyword parameters" often found in command languages.

A list of values is a series of entries of the form `A=B`, in which the entries are separated by 001 (control A) characters. Each substring consists of a parameter name, an "=" character, and then the actual value. For example, the *Data* string for a "write to file" opcode might be

```
HANDLE=1234^ACOUNT=13^AOFFSET=0^ADATA=Test message\n
```

This string defines values for the parameters `STREAM`, `COUNT`, `OFFSET`, and `DATA`.

A list of values is not a set of values: a parameter can appear more than once in the list.

3.2. Printserver Protocol

The client communicates with the printer using the *printserver protocol*. This is a simple text-oriented protocol layered on top of TCP. The server sends a counted record to the printer, and the printer replies with counted records in return. Some server-to-printer messages require replies, and some do not.

Here are the Opcodes and their symbolic names:

0		Null. Do nothing. No reply is expected, and all of the fields are ignored.
1	ssn	Begin a new print session. Initialize everything. Reply required. The *Data* array is a list of values, as follows: • `SESSIONID` is the host system's identifying string (job number, etc.) • `HOST` is the string name of the the relay host. This is the name of the host making the actual TCP connection to the printer. • `NOTE` an uninterpreted character string that might contain information useful to a person looking at the printer or at the output. The reply is a list of values containing information about the server and the session number, as follows: • `SERVERJOBNUMBER` the string name assigned by the server to this print session, which could, for example, be used at the console to enquire about the job and to abort it. • `SERVERID` a string identifying the version of the software running in the server; its only use is for information. • `PRINTERHOST` the network host name of the printer; this can differ from the name of the printer. If the server is not willing to accept the connection, as for example it has not finished its initialization, it should respond with a *nak* opcode rather than by rejecting the connection or by waiting until it is ready. There is no "End of session" command; rather, the network connection is closed after an *eoj* command (see below) to indicate the end of a session.
2	wait	Wait for the completion of a print session. The client is not obligated to wait, but if it closes the connection it will not receive error messages about paper jams and the like. Error messages that originate in the PostScript interpreter will be sent back to the relay server before the *eoj* is replied to, but error messages that result from the paper transport mechanism can still be produced by the printer even after a job has ended. If a *wait* command is issued, then when its reply is received, the client can be confident that every page that has been imaged has also been landed successfully in an output stacker.
3	soj	Start of job. A "job" is a sub-part of a "session". All of the jobs in a session come from the same client host, and will be printed together without interruption, but each job is printed in a clean copy of the PostScript interpreter. Reply not required. When the client sends an *soj*, it may start sending data immediately without waiting for a reply. If the printer is not prepared to handle the job data, it will block the transmission at the network level and the client will hang on its "send" command until the printer permits transmission to resume.
4	eoj	End of job. This command indicates the end of a print job. There will be a reply whose data array is a list of values, one of which will be named PAGES. After an *eoj* command, either the client should close the network

		connection (indicating end of session), a *wait* command should be issued (indicating that the client would like to keep the link open to receive possible error messages but is otherwise done with the session), or a new job should be started with an *soj* command. The difference between the receipt of the reply to an *eoj* and the receipt of the reply to a *wait* that follows it is that the reply to the *eoj* can be sent when the last page has been imaged, while the reply to the *wait* cannot be sent until the last page has properly cleared the paper stacker without jamming.
5	data	PostScript data. Reply not required.
6	kill	Kill current print job and session. This opcode is generated by the client software in response to a client-side abort request. The client is encouraged to send a *kill* opcode as TCP urgent data; however this opcode is sent, the client is expected to follow it with a *wait* opcode (if desired) and then close the connection immediately afterwards. The client can then assume that the print session is ended. The server will send the client a *kill* opcode in the event of unrecoverable server failure, but not in the event of premature termination of an ordinary print job.
7	info	Provide user-identification information to the server. The *Data* array is a list of values, as follows: • `USERID` is the string name of the user whose file is being printed. • `SESSIONID` is the name of the print file or print session. • `HOSTNAME` is the name of the host computer from which user `USERID` printed the file. • `NOTE` is a commentary that will appear on the server's console and in its log files. On some systems, the print jobs of multiple users are batched together by the line printer daemon into a single session. When this happens, the client software sends the server an *info* packet whenever the user identification changes. Sometimes the components of the user identification are not all changed at once; any field whose new value is not yet known should be left unspecified in an *info* command.
8	eof	Some implementations of TCP have difficulty handling an unannounced closing of a TCP connection. The *eof* opcode is an advisory-only packet that, if sent, is a notification to the server that the next action on that connection will be to close it. The *eof* opcode may be completely ignored on any system with a properly-working TCP implementation.
9	flush	Some implementations of TCP can be run more efficiently by batching data records into larger blocks, to reduce the number of system calls. The client should be free to do so. If clients are batching data records, they are advised to send a *flush* opcode at the end of each print job, to assist the server in recognizing the end of the job. The server is free to ignore this opcode if it so wishes. In general the *flush* opcode will be useful only if the server is actually a network relay to the true server.
101	repl	Reply. This record is a reply to the record whose *ID* it shares. The contents of the *Data* array depend on the command being replied to. If the text of the

		reply requires more than one record, then all but the last record must have opcode 102, *prepl* (see below). *ssn*: The reply to a *ssn* command has a of values consisting of the single value `SESSIONID`, which is the name of the session, as assigned by the print server. Typical names are "`1`" or "`2`" or "`13`". *eoj*: The reply to an *eoj* command contains the counts of the number of pages and/or images printed by this job. The reply is a list of values. *time*: The reply to a *time* command contains the current time. The format is explained in the documentation of the *time* command.
102	prepl	Beginning of a multipart reply. If the amount of text for a reply (e.g. for a "show queue" command) is too big to fit in one record, then all records but the last must have opcode 102. When a 101 reply is received, the reply is taken to be complete.
103	nak	This is a "negative reply". The recipient of a record has rejected the request made by the sender. The sender is expecting a *repl* opcode, but if it receives a *nak*, it will withdraw its request. The *Data* array is a text string explaining the reason for the rejection of the request.

3.3. Printer management protocol

In addition to client connections to the print server for the purpose of sending data, it is also possible to connect to the print server for the purpose of sending management information. The print server does not accept management information over a data connection because of the possibility of fraudulent data.

The print server is passive. It does not ever make connections. However, it will not print until it receives a connection from a master management client, so that it can read its configuration and startup data. Since the printer does not make connections, it must wait for the management client to make a connection to it. If an ordinary client tries to begin a session before the printer has been contacted by a master management client, that session will block until the configuration and startup have completed.

The responsibility of finding the print server, knowing its network address, and making a connection to it lies with the client management software. It is expected that a client host will run a daemon process that will maintain a connection to a running print server at all times, and will probe at relatively short intervals (every 30 seconds or so) when it loses contact.

Once the management client has established a connection with the printer, the printer becomes the active agent and the management client becomes passive. The printer will ask the management client for certain services and receive replies; it will also send the management client accounting and logging information. Requests from the printer to read and write files will be directed only to one master management client.

The accounting information that is sent to the management client is separate from the accounting information stored on the individual client hosts. Each client will receive accounting information about the files that its users printed on the printer; the management client will receive accounting information about all printing, regardless of its origin.

There can be multiple management clients to receive management data, and all will be sent accounting data. While there can be multiple master management clients, only one of them will actually be used as a master; the rest will merely receive accounting information as if they were normal management clients.

The exchanges between a print server and management clients use precisely the same syntax and protocol as is used for printing files from ordinary clients. There are a few additional opcodes that are recognized only from a management client, and there are a few additional opcodes that are sent only to management clients.

Although a newly-booted printer cannot function until it has made contact with a master management client and has received and processed the access and configuration information, the printer will continue to function even if the master management client crashes, as long as there is at least one management client connected and able to receive accounting data.

Here are the management opcodes and their symbolic names:

41 mssn Begin a new management session. Similar to the *ssn* opcode, except that it begins a management session rather than a client session. The *Data* array contains a list of values providing some required information and then naming the services that the management client is willing to provide to the PrintServer. Required parameters:

- `PASSWORD`: the PrintServer is free to reject the management session request if it does not like the value assigned to PASSWORD.
- `HOST`: the host name requesting the management connection. PrintServer is free to reject the management session request if it does not like the value assigned to HOST.
- `PRINTERHOST`: the network host name of the printer itself. Initially it knows only its address.

The remaining parameters are all optional. To declare that it is willing to offer a certain service, the management client sets the value of the service name to 1. For example, if it is willing to provide the CFREAD service (read access to configuration files), then the string `CFREAD=1` should appear in the list of values. The specific service code names are not part of the protocol, but rather are part of a private agreement between the server and its management client(s). Typical values might include:

CFREAD	Read access to configuration files
FILEIO	Read/write access to certain file directories.
ACCOUNT	Receives and stores accounting data.
ERRLOG	Receives and stores error and log data.

The management client must wait for the server to reply.

42 time Request the current data and time. A management client is expected to reply to a *time* opcode with a *reply* record whose *Data* part is the current time. The *Data* array of the reply contains a character string specifying the current local time, in the form

```
dd-mmm-yyyy hh:mm:ss
```

where

dd is the 2-digit day of the month, 01 to 31
mmm is the 3-letter month code, JAN to DEC.
yyy is the 4-digit year, 1971 or later
hh is the 2-digit hour, 00 to 23
mm is the 2-digit minute, 00 to 59
ss is the 2-digit second, 00 to 59

The date and time string may have leading or trailing spaces, but the content portion must be exactly 20 characters. The month code is the first 3 letters of the English spelling of the name of the month, and can be in any combination of upper and lower case letters.

43 acct Sent by the print server to all management clients that offered `ACCOUNT` service when they connected, asking the client to record accounting information. The *Data* array passed to the management client by the printer is a list of values which should be written to the accounting file. Not every *acct* command will define every value; the recommended behavior of the management client is to write the entire *Data* array to the accounting file, intact, and to let a postprocessing program worry about decoding the several possible combinations of parameters. The parameters that can be defined include:

- `USER`. The name of the user who printed the file, as passed to the printer by the most recent *info* command.
- `HOST`. The network name of the computer from which the file was printed, as passed to the printer by the most recent *info* command.
- `PAGES`. The number of pieces of paper consumed by the print job.
- `IMAGES`. The number of images produced by the PostScript interpreter. This value can differ from `PAGES` in the case of 2-sided printing.
- `TIME`. The number of seconds that the printer was unavailable for other use because of its attention to this print job.
- `TRAY`. If the printer has more than one paper tray, the TRAY parameter names the paper tray that was used. If a job prints 5 pages on Tray1 and then 3 pages on Tray2 and then 2 more pages on Tray1, the list will say `TRAY=Tray1`, then `PAGES=5`, then `TRAY=Tray2`, then `PAGES=3`, then `TRAY=Tray1`, then `PAGES=2`.

The management client must send back to the PrintServer a reply acknowledging receipt of the accounting data.

44 emsg Error message. Nearly identical to the *data* opcode, except that its contents are an error message pertaining to the status of the PrintServer rather than to the status of the job that the user is printing on the PrintServer. These error messages will be sent to all management clients that offered `ERRLOG` service when they connected.

45 cssn Console session. Used to begin a "remote console" session. A management client that connects for a "remote console" session can exchange *data* records with the printer in the format prescribed in the companion *Server Management and Accounting* document. The *Data* array of a *cssn* command is a list of values, as follows:

- `USER`: the logged-in name of the user requesting the remote console session.
- `HOST`: the hostname on which the user is logged in.

The print server will reply with a a series of *data* opcodes, at intervals, which will draw certain information on the screen. The server and client exchange *data* records with one another until the *cssn* is terminated by either end closing the connection. The format of these *data* records is not part of the Print Server Protocol, but is rather part of the implementation-specific nature of the server code itself.

The next 4 opcodes are used to give the PrintServer access to a file system. If the management client has offered `CFREAD` *service, then it must be prepared to handle* open, read, *and* close *opcodes, and it must know how to find the configuration and setup files from the names* `$CONFIG` *and* `$SETUP` *used as arguments to* open. *If the management client has offered* `FILEIO` *service, then it must also be prepared to handle the* write *opcode, and it must have a file name space (usually a directory) available in which it can read and write files as directed by* open *commands.*

50 open Sent by the print server to the management client, asking the client to open a file and send a reply back to the printer containing the file number. The *Data* array of the *open* command is a list of 2 values:

- `PATH`: a file name, which the management client is free to interpret however it wishes. Normally the management client will add a prefix to the beginning of the file name to restrict its domain, e.g. if the printer asks to read file "/etc/passwd", the management client will open some file like "/usr/local/lib/lps/rdisk/etc/passwd", if it exists.
- `TYPE`: A string describing how to open the file. If the string contains the letter "r", then the file will be opened for input; if it does not exist, then an error status will be returned. If the string contains the letter "w", then the file will be opened for output; if it does not exist it will be created, and if it does exist, then the old copy will be overwritten.

If `PATH` is equal to "`$SETUP`", then the file that will be opened is the PostScript setup file for this printer, regardless of whether or not there is a file named "`$SETUP`". If `PATH` is equal to "`$CONFIG`", then the file that will be opened is the configuration parameter file for this printer, regardless of whether or not there is a file named "`$CONFIG`".

The reply sent back to the printer will be a list of values:

- `RETURN` is the file handle that must be used by the printer in future "read", "write", and "close" commands to the management client.

- `ERROR`, if present, means that the request to open a file has failed; the value of `ERROR` is the text of the error message.

It is not permitted to open a file named `$CONFIG` or `$SETUP` for output; this is to prevent the configuration and setup files from being overwritten.

52 read

Sent by the print server to the management client, asking the client to read from a file and return the data so read back to the printer. The *Data* array passed to the management client by the printer is a list of values:

- `HANDLE`: a file handle, as returned from a call to *open*
- `OFFSET`: the offset, measured in bytes from the beginning of the file, at which to begin the read. To read the second 512-byte block from a file, set `OFFSET=512` and `COUNT=512`.
- `COUNT`: the integer count of the number of bytes to read from the file.
- `LINE`: if the value `LINE` is defined, then (regardless of its defined value) the data returned back to the printer will never be more than one line. A "line" is defined to be the characters beginning with `OFFSET` through and including the next newline (control-J, octal 12) character in the file. IF a *read* operation stops at a newline character because of the `LINE` option, then that newline character will always be the last character in the returned data.

The count must not exceed 512. The management client will respond by sending back to the printer a reply containing a list of values:

- `RETURN` is the number of bytes actually read from the file. It will be equal to the incoming value of `COUNT` unless the file was too short or unless the `LINE` value was specified and a newline character was seen.
- `ERROR`, if present, means that the *read* request has failed. The value of `ERROR` is the text of the error message.
- `DATA` is the data actually read from the file

The name `DATA` will be the last name in the list of values, and it can contain any character at all, including the 001 (control A) character normally used to delimit fields in a list of values. The length of the `DATA` string is determined by the `RETURN` value.

53 write

Sent by the print server to the management client, asking the client to write to a file from the data passed to it. The *Data* array passed to the management client by the printer is a list of values:

- `HANDLE`: the file handle, as returned from a call to *open*.
- `OFFSET`: the offset, measured in bytes from the beginning of the file, at which to begin the write.
- `COUNT`: the count of the number of bytes to be written, which must not exceed 512.

- `DATA`: is the block of bytes to be written. It must be the last value in the list.

The block of bytes to be written can contain any character at all, including the 001 (control A) character normally used to delimit fields in a list of values. (This is why the block of bytes to be written is the last value of the list.) The reply sent to the printer by the management client is a list of values:

- `RETURN`: the number of bytes actually written.
- `ERROR`: if present, then the *write* request has failed and the value of `ERROR` is the text of the error message.

54 close Sent by the print server to the management client, asking the client to close a previously-opened file. The *Data* array passed to the management client by the printer is a list of values:

- `HANDLE`: the file handle to be closed.

The reply sent back to the printer will be a list of values:

- `ERROR`: if defined, then the *close* request has failed and the value of ERROR is the text of the error message.

4. Examples

To print one PostScript file, an ordinary client should use the PrintServer protocol as follows:

Client sends	**Server response**	
SSN		*begin a session*
	REPLY	*acknowledge it*
INFO		*identify the coming print job*
SOJ		*start a print job*
DATA		*send the file*
...		
DATA		
EOJ		*end the print job*
	REPLY	*Return page count*
WAIT		*wait for all error messages*
	REPLY	*announce the end*

To print a break page and then two PostScript files, a client should use the PrintServer protocol as follows:

Client sends	Server response	
SSN		*begin a session*
	REPLY	*acknowledge it*
INFO		*identify the coming print job*
SOJ		*start a print job*
DATA		*send the break page*
...		
DATA		
EOJ		*end of break page*
	REPLY	*Return page count*
INFO		*identify the coming print job*
SOJ		*start a print job*
DATA		*send file 1*
...		
DATA		
EOJ		
	REPLY	*Return page count*
INFO		*identify the coming print job*
SOJ		*start a print job*
DATA		*send file 2*
...		
DATA		
EOJ		
	REPLY	*Return page count*
WAIT		*wait for all error messages*
	REPLY	*announce the end*

Here is the exchange that takes place if the client begins printing a file and then decides to kill it:

Client sends	Server response	
SSN		*begin a session*
	REPLY	*acknowledge it*
INFO		*identify the coming print job*
SOJ		*start a print job*
DATA		*send the file*
...		
DATA		
KILL		*end the print job*
	REPLY	*Return page count*
WAIT		*wait for all error messages*

Table of Contents