Note: This is a preliminary first draft version of this document. Portions may be missing or subject to change in the future. This document is intended as a starting point for developers who wish to work developing clients or servers for this protocol. Developers who are not interested in either of these two areas can ignore this document.

Direct questions and comments concerning this document to: DEVSUPPORT

Apple Printer Sharing Protocol Description

This document is a specification for the Apple Printer Sharing Protocol (APSP) Version 1.0. APSP is a server/client connection-based stream used for sending documents to print servers. APSP is intended for use on Macintosh computer using the AppleTalk Data Stream Protocol (ADSP) transport. However, the protocol was designed with portability in mind. APSP requires only that the underlying transport guarantee point-to-point connections, reliable delivery, and correctly ordered messages. Implementing APSP over a non-ADSP transport, or even in a non-Macintosh environment, should not prove difficult.

The protocol exchange consists of transaction queries and responses, and bulk data transfer. APSP is best implemented using two channels, one for transactions and one for bulk data exchange. The description for each message describes the input and output parameters, possible errors, and the effects of the given message for the client and server.

Data Types

The following definitions are used to describe call parameters:

char	1-byte quantity (Macintosh character)
short	a 2-byte quantity
long	a 4-byte quantity
boolean	a short, TRUE if 1 (one), FALSE if 0 (zero)
bitfield	one or more unsigned shorts
Str31	length char, 0 to 31 char’s (max 32 bytes)
Str32	length char, 0 to 32 char’s (max 33 bytes)
Intl31	script word, length byte, 0 to 31 char’s (max 34 bytes)
DESBlock	32-byte unsigned quantity

Protocol Conventions

APSP uses the following conventions:

• All integral types are signed values unless otherwise indicated.

• Bit and bytes are ordered as per the MC680x0 processor family.

• All strings use the Macintosh Extended ASCII character set and the Macintosh international script code(s).

• All variable-length parameters (i.e. strings) are null padded to a word boundary.

In particular, NULL strings are two-byte quantities: a length byte of zero, followed by a NULL pad byte. The length field of the message will include the pad byte(s), though the string length byte will not.

• Response packets with non-zero error codes do not contain any additional parameters, unless otherwise noted.

Transaction Message Format

APSP conceptually uses two channels: a transaction channel, and a data channel. The transaction channel is designed for request/response messages, and is used mainly for session control and status. Figure 1 shows the format for a transaction message.

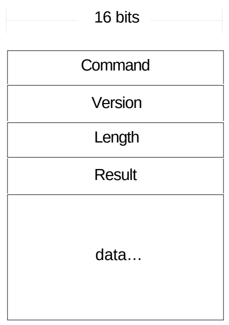


Figure 1. Transaction Message Format

The header portion of the message contains:

• APSP Command code (short)

• APSP Version code (short). An APSP version is actually two packed bytes. The ms byte is the major version; the ls byte is the minor version. Ex: 0x0103 is version 1.3. This document describes version 1.0 (0x0100).

• Message Length (short). Number of bytes in the message *including* the header, i.e. 8 + data length.

• Result code (short). This field is ignored for requests.

The data portion of the message contains transaction-specific parameters. See the message descriptions below. Only one transaction can be outstanding at a given moment. Any client request after the first will be ignored by the server.

Note that this implies a one-transaction limit for the client API. A transaction is started when the client makes a request. The transaction is completed when the client receives a reply, or the transaction is aborted.

Data Message Format

The data channel is used for transferring bulk data (print jobs and job format negotiation). Figure 2 shows the format for a data message.

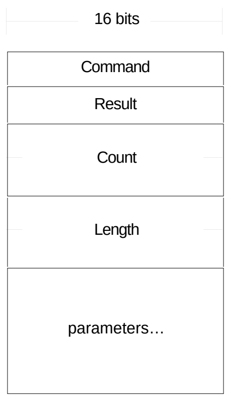


Figure 2. Data Message Format

The header portion of the message contains:

- • APSP Command code (short)
- • APSP Result code (short)
- • Parameter count (long)
- • Message Length (long). Number of bytes in the message including the header, i.e. 12 + data length.

The parameter portion of the data message contains a list of parameters. Each parameter has a format given by Figure 3.

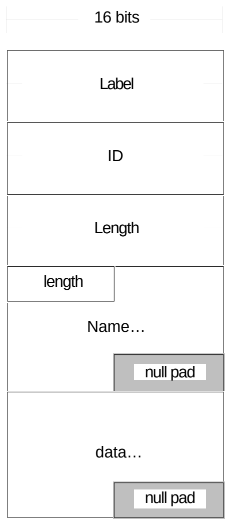


Figure 3. Paramater Format

Each parameter contains:

- • parameter label (unsigned long)
- • parameter ID (long)

- parameter length (long) number of bytes in the parameter including the parameter header, i.e. 12 + name length (+ pad) + data length (+ pad)

- parameter name (Str255, null padded to word boundary)

- parameter data (arbitrary bytes, null padded to word boundary)

A complete description of the data messages can be found in Bulk Data Formats (see below).

APSP Over ADSP

APSP is designed to operate over ADSP. Transaction messages are sent and received via the Attention Channel. Data messages are sent and received over the Data Channel. All transactions are sent as attention packets, with the attention code set to the APSP message command number

ADSP limits a single attention message to 570 bytes. When sending an APSP message that is longer than 512 bytes of data, the sending side will set the result code to psNotDone, indicating that the message is larger than a single ADSP attention packet, and the receiver should expect more packets. The sender will set the APSP result code of the last packet to something other than psNotDone when the entire message has been sent. In other words, an n-packet message will have n-1 packets with a result code of psNotDone, and the last packet will have the actual result code. The header data is the same for each packet (with the exception of the result code). The message data can be split into packets of any size up to 512 bytes, as long as all bytes of the message are delievered in sequence.

Data packet ends are marked by the ADSP EOM flag, as well as the message length field. In some of the data packets, the message length field is filled with psNoLength (-1) indicating that the length of a particular packet or parameter is unknown. In this case, the server or client would need to parse the data by walking the parameters to determine the length, or rely on the EOM bit to mark the end of the packet. Notice that in the Macintosh implementation, packets which use the psNoLength value are **required** to set the EOM bit to mark the end of the packets.

APSP Over Other Transports

Other transports may be used so long as the transport guarantees reliable delivery and correctly-ordered messages. The APSP implementation must adhere to all APSP specifications, even if the transport does not impose the same restrictions as ADSP (i.e. only one outstanding transaction). Note that APSP “transaction” and “data” channels are conceptual entities. A single channel implementation is feasible assuming the client and server have someway to distinguish between transactions and data messages. Other implementations may provide out-of-band data to coordinate multiplexing transaction and data packet streams.

Magic Cookies

APSP tries to minimize bandwidth usage by using tags called magic cookies. A magic cookie is guaranteed to change uniquely when the data it represents changes. The server uses the tag to determine if a client has the most recent information. A server may return a special value meaning “no change” for certain requests.

> For example, suppose a client requests the name of the currently printing document. The server would reply with the name and a magic cookie. The client includes the magic cookie the next

time it requests the current document’s name. The server can then compare magic cookies and respond with “no change” if the same document is still printing, or the name of the new document (and a new magic cookie).

A magic cookie is respresented as a 32-bit signed quantity. A magic cookie of 0 is never considered to be current; i.e., a client can send a magic cookie of 0 to force the server to respond. The value of -1 indicates that the server does not support magic cookies for that item.

To Image Or Spool?

Before transferring a job to a server for printing, the server and client must agree on the job format. Specifically, the client and server must decide whether the client should send a spool file or an image file. Determining when to remote image and spool is a complex process (referred to as “the negotiation”). The following set of rules should determine when to locally or remotely image:

1) Does the client request to send an image job?
   If yes, remote spooling, if no, continue.
2) Does the server request only an image job because of any of the following reasons?
   • version mismatches (server version < client version)
   • machine type mismatch (server machine is much slower than client)
   • server is set up to only handle image jobs
   If yes, remote spooling, if no, continue.
3) Did the server request any needed fonts or extensions?
   If no, remote imaging, if yes, continue.
4) Can the fonts be sent?
   If yes, send and remote image, if no, continue.
5) Are the fonts and PerMods required?
   If yes, inform user of error, if no, warn user and continue.

Bulk Data Formats

Job format negotiation and submission take place on the data channel. APSP uses five data messages: Requirements, NeedResources, Spool, Resources, and Image. All five have the same format, given in Figures 2 and 3 above. If the job is being locally imaged, job transfer can begin immediately using Image messages with ‘imag’ parameters.

If the job is being imaged remotely, the client sends the server a list of required resources (fonts and printing extensions) using one Requirements messages with ‘font’ and ‘ext ’ parameters. The client sends one parameter for all of the fonts, and another for all of the printing extensions needed to image the job. The server responds by sending one or more NeedResources message, detailing the required resources that the client must supply, i.e. a font which the job uses that is not installed on the server. The client then has the option of supplying the resource(s) in one or more Resource messages, imaging the job locally and sending an image job, or cancelling the job transmission. The server assumes that the client will be responsible for sending the needed resources, or imaging the job in such a way as to not require them.

Parameters of Bulk Data Packets

Here are a description of the parameters used within these data messages, listed as type, ID, and name. In addition, each one has an indication of where it is considered legal to use, and what it means.

‘spol’ (0, “”)— this data is in the form of a series of flattened pictures, one per page. This is the same data format as the data fork of a disk spool file in GX. Expected to be first parameter in packets of type psSpool. You must supply either this parameter, or ‘imag’ when sending a psSpool packet.

‘imag’ (0, “”)— this data is driver specific, but consists of a standard header, plus standard tagged image format consisting of tag (long), length (long), data (variable). A series of these will be placed together to define data for the printer, page boundaries, and printer queries. The Image File Format will be discussed further in another document. Expected to be the first parameter in packets of the type psSpool. You must supply either this parameter, or ‘spol’ when sending a psSpool packet.

‘font’ (0, “”) — a series of required (or missing) font names, in the form of Pascal strings appended together. Legal only in psRequirements and psNeedResources packets.

‘ext ’ (0, “”) — printing extension information in the form of:
```
        OSType          creatorType;                    // type of the extension
        long            version;                        // version of the extension
        long            flags;                          // flags (where to use) for the extension
```
per extension. Legal only in psRequirements and psNeedResources packets.

‘XXXX’ (ID, NAME) — any Macintosh resource, sent to be placed into the resource fork. Legal only in packets of type psResources.

Expected use of data packets

After receiving a “noErr” from a SendJob request, the server and the client are expected to be in the process of sending a file. This sending takes place entirely with the use of data packets and proceeds as follows.

1) Client sends the server a packet of type psRequirements.
2) Server replies with a packet of type psNeedResources.
3) Client sends to the server one of the following:
    - A psSpool packet consisting of ‘spol’ data. Following should be a psResources packet containing the resource fork of the spool file, plus any additional fonts needed by the file.
    - A psSpool packet consisting of ‘imag’ data. Following should be a psResources packet containing the ‘job ’ and ‘frmt’ resources.
    - an AbortJob message
4) Assuming the job has not yet been aborted, the server then sends the client a packet of type psSpool with no parameters, containing a final error code for the transmission.

Security Model

APSP security is made up of three components: user class, authentication type, and access rights. User classes allow the server administrator to provide the users with access to a subset of the server functionality. The server uses authentication types to verify that the client’s user class. Access rights determine the services provided to the authenticated client.

User Class

APSP supports five classes of user:

Guest
Anonymous client. No password is required.

Zone Guest
Anonymous client located in the server's local zone. No password is required.

User
A client presenting a user name. A password may be required.

Group
A client presenting a user name enumerated in a group of users registered for the server. A password may be required.

Administrator
A client presenting the user name designated as the administrator for the server. A password may be required. This user has by definition all privileges. An APSP server is not required to designate an Administrator.

Authentication Type

PrinterShare supports six types of authentication:

uamGuest
Guest access. No verification is performed by the server.

uamZoneGuest
Zone membership. The server verifies the client's zone membership.

uamNoPassword
User name, no password. Treated as uamGuest + uamZoneGuest.

uamClearText
User name, password in clear text.

uamRandNum
User name, password encrypted using AppleShare DES. Passwords are null-padded to 31 characters before encryption or decryption, making the key and data 32 bytes long.

uamTwoWay
User name, password encrypted using AppleShare DES, server validation. Passwords are null-padded to 31 characters before encryption or decryption, making the key and data 32 bytes long.

Access Rights

APSP access rights determine what level of access of client has to server resources:

privNone	client has no priveleges
privConnect	client may connect to the server
privChangePassword	client may change user's password

privSendJob	client may submit jobs
privGetOwnJobInfo	client may request information about user jobs
privSetOwnJobInfo	client may change information about user jobs
privDeleteOwnJob	client may delete user jobs
privGetGroupJobInfo	client may request information about group jobs
privSetGroupJobInfo	client may change information about group jobs
privDeleteGroupJob	client may delete group jobs
privGetOtherJobInfo	client may request information about other jobs
privSetOtherJobInfo	client may change information about other jobs
privDeleteOtherJob	client may delete other jobs
privSetQueueState	client may change queue state (start/stop printing)

Note: *User job* is any job the user (client with the same user name) has submitted. *Group job* is any job submitted by a member of the user's group(s). *Other job* is any job submitted neither by the user, nor by a member of the user's group(s).

How Security Works

A client must choose one of the six authentication types (authTypes) when submitting an Authenticate request to an APSP server. The authTypes above are ordered in "increasing" security, judged by the amount of work the server and client must do to complete the authentication process.

For example, a client requesting guest access requires no additional information, and no verification on the part of the server. A client requesting group access must present a user name and password, and the server is required to verify that the password is valid and the user is a member of a valid group.

The authType is an indication to the server as to the user class of the client. The client could belong to more than one class (registered user and zone guest, for instance). The server must determine all possible client user classes given the client's authType.

Access rights are assigned to each user class. The server will grant all access rights defined for the client's user class(es). In other words, a client that is identified as both a guest (any user) and a zone guest (any user in the same zone as the server) will be granted access rights for both guest and zone guest.

The section below describes how the server determines a user class based on the client's authType, and notes which access rights are granted to an authenticated client.

uamGuest
If the server supports guest access, return guest access rights. If not, return "no guests."

uamZoneGuest
If the server does not support zone guests, return "no zone guests." If the server supports zone guests and the client is in the server's local zone, return zone + guest access rights. If not, return "wrong zone."

uamNoPassword
Treat as uamZoneGuest + uamGuest, except the client must specify a user name. Note that the server is not required to verify that the user is who she says she is. Clients may use this authType

to avoid requesting a password every time the user wishes to access the server.

uamClearText, uamRandNum, uamTwoWay
The server authenticates the client in several steps:

	Defined User	Valid Password	Registered User	Registered Group	Local Zone	
Case 1	N	*	*	*	*	reject: no such user
Case 2	Y	N	*	*	*	reject: bad password
Case 3	Y	Y	N	N	*	treat as uamNoPassword; reject: no such user
Case 4	Y	Y	N	Y	N	return group + guest
Case 5	Y	Y	N	Y	Y	return group + zone + guest
Case 6	Y	Y	Y	N	N	return user + guest
Case 7	Y	Y	Y	N	Y	return user + zone + guest
Case 8	Y	Y	Y	Y	N	return user + group + guest
Case 9	Y	Y	Y	Y	Y	return user + group + zone + guest

Notes:
Defined User - user exists, but is not explicitly configured for access
Registered User - user exists, and is explicitly configured for access
Registered Group - group of Defined Users
Local Zone - the client is located in the server’s local zone

CloseSession

The client advises the server that it wishes to gracefully terminate the session. The session will not be closed if the session is still “busy.” A session is considered busy if a SendJob call is in progress (see below).

Required Priveleges

privNone

Request Parameters

none

Response Parameters

none

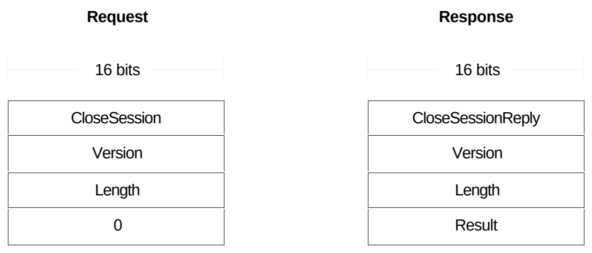
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psSessInUse	session is in use and can’t be closed

Algorithm

The connection is closed by the server and client at the end of receipt of the reply. It is an error to transmit either data or transactions after positive acknowledgement of a CloseSession.

Message Format



Authenticate

The client wishes to authenticate itself to the server, and thereby gain any special access rights configured for the user.

Required Priveleges

privNone

Request Parameters

uam	which UAM to use [unsigned long]
userName	user name [Str31]
password	user password [Str31]
nbpName	client registered NBP name [Str32]
nbpType	client registered NBP type [Str32]

Response Parameters

privs	client priveleges after authentication [unsigned long]
authID	authentication ID, used by Validate [long]
randNum	random number, used by Validate [DESBlock]

Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psBadUAM	server doesn't grok this UAM type
psBadParm	userName, password, nbpName, nbpType length is invalid
psNoGuest	guest access not allowed
psNoZoneGuest	zone guest access not allowed
psWrongZone	client is not in server's local zone
psNoSuchUser	userName is unknown
psBadPswd	password is wrong
psNotYet	client must submit a Validate request (see below)

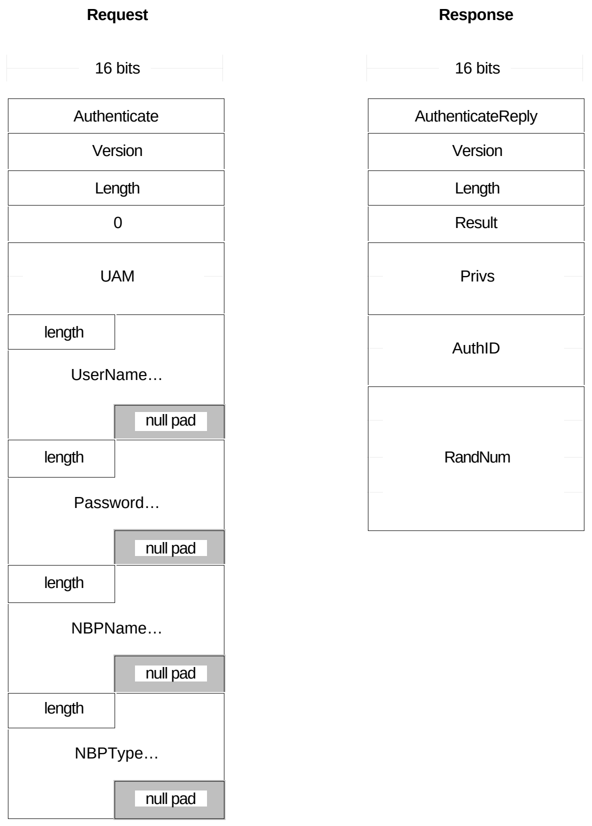
Algorithm

The authentication process is described under Security Model above. A client may authenticate a session at any time. A client may authenticate as many times as desired. The authentication process may cause a client's access rights to increase or decrease, depending on the server configuration and the UAM. All request parameters are required to be present, even if they are not used for the particular UAM requested by the client. Unused parameters should be set to NULL:

uamGuest – null userName, password, nbpName, nbpType
uamZoneGuest – null userName, password
uamNoPassword – null password
uamClearText – (optional) null nbpName, nbpType
uamRandNum – (optional) null nbpName, nbpType
uamTwoWay – (optional) null nbpName, nbpType

The authID and randNum parameters will only be returned for uamRandNum and uamTwoWay, in which case the response result code will be psNotYet, and privs will be set to privNone.

Message Format



Validate

A server may request a client to validate an authentication request by presenting password credentials.

Required Priveleges

privNone

Request Parameters

authID	ID number returned by Authenticate response [long]
encryptedPass	random number encrypted with client password [DESBlock]
randNum	random number [DESBlock]

Response Parameters

authID	ID number returned by Authenticate response [long]
encryptedPass	client password, encrypted [DESBlock]

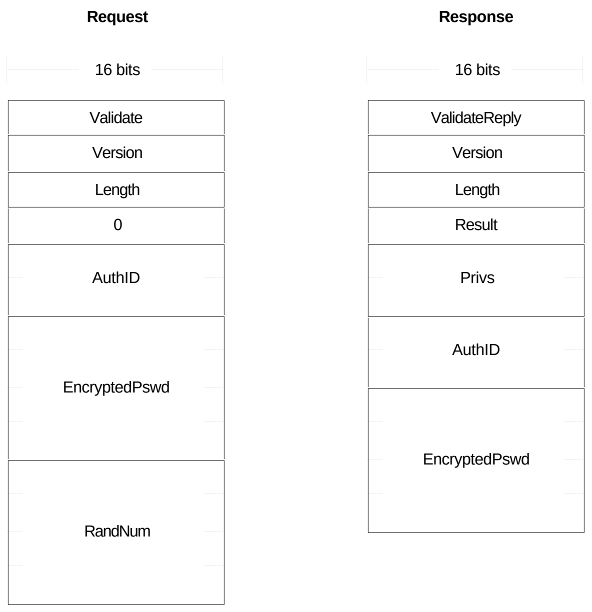
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoAuthInProg	Validate without prior Authenticate
psBadAuthID	authID is invalid
psBadParm	required parameter missing
psBadPswd	password is wrong
psNotYet	user must submit a(nother) Validate request

Algorithm

Certain UAMs (Random Number Exchange, and Two-Way Scrambled) require the server to validate the client before authnticating the session. The client encrypts the random number returned by the server in Authenticate using the client's password (Str31 null-padded to 31 characters) as the encryption key. The client sends the encrypted random number along with the authID returned by the server in Authenticate. For Two-Way Scrambling, the client may also request the server validate itself by sending an additional random number, which the server encrypts using the client password and returns in the Validate Response. If the client determines that the server is not valid (i.e. does not know the client's password) the client must initiate a CloseSession. Note that future authentication schemes may require several Validate transactions before authenticating the session.

Message Format



ChangePassword

Request the server to change the password for a given user. This message does not require the client session to be authenticated. The required priveleges are determined based on the request parameters sent by the client.

Required Priveleges

privChangePassword

Request Parameters

UAM	which UAM to use [unsigned long]
userName	username [Str31]
oldPassword	old password [Str31]
newPassword	new password [Str31]

Response Parameters

none

Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psBadUAM	server doesn't grok this UAM type
psBadParm	userName, oldPassword, newPassword length is invalid
psNoPriv	no privelege for attempted operation
psBadPswd	oldPassword is wrong

Algorithm

If the UAM is uamClearText, the client sends the server its user name plus its old and new passwords in clear text. The server looks up the password for that user, if it matches the oldPassword parameter, the newPassword will be saved for that user.

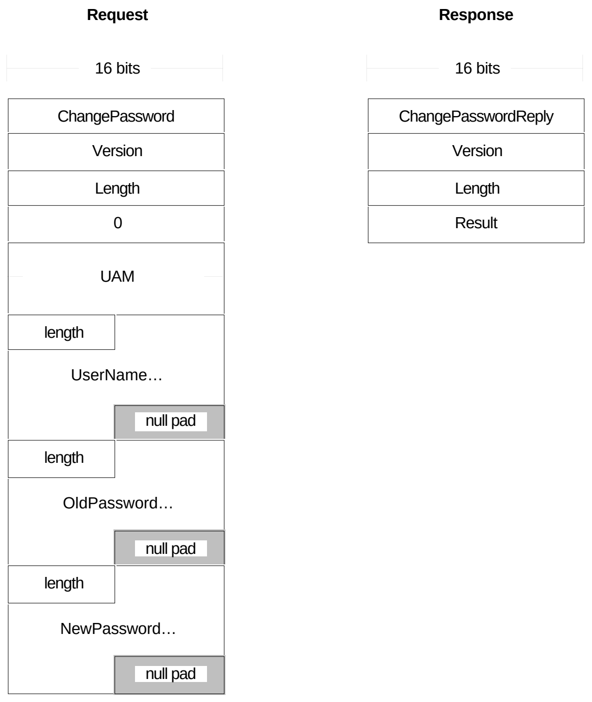
If the UAM is uamRandNum, the client sends the server its user name, the user's old password DES encrypted using newPassword as the key, and the user's new password DES encrypted using oldPassword as the key. Both passwords are Str31's null-padded to 31 characters. The server looks up the password for that user, uses that password as the key to decrypt the new password, and uses that result as a key to decrypt the old password. If the final result matches the server's copy of the old password, then the new password will be saved for that user.

All other UAMs are considered to be invalid (i.e., will return a psBadUAM result code).

Notes

The granting of the ability to change a password is an administrative function and is outside the scope of this protocol specification. Servers may optionally choose not to implement this function.

Message Format



SendJob

This request comes from the client to request the transmission of a job to the ImageStation.

Required Priveleges

privSendJob

Request Parameters

imageLocally	client wishes to send an image job [boolean]
driverVersion	driver version to use [long]
tackleboxVers	version of Print Manager]long]
machineType	client machine ID (Gestalt) [long]

Response Parameters

imageLocally	client should send image job [boolean]
queueID	identity of queue [long]
deviceID	identity of device [long]
jobID	identity of job [long]

Result Codes

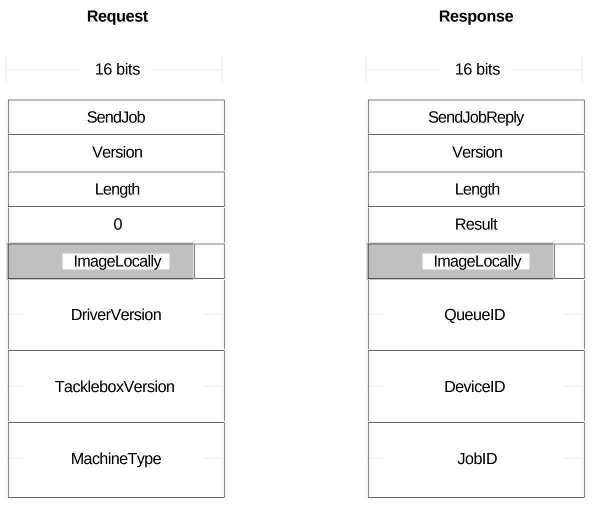
psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation

Algorithm

This protocol command starts the transmission of a job. The server returns to a jobID for use in future packets concerning this job.

If the result code is noErr, the client may begin Job Negotiation as described above. The jobID should be used for any requests (like GetJobInfo) the client may make concerning the job. If the server returns "noErr" from this call, it expects the client to begin the sending of data packets, begining with psRequirements.

Message Format



AbortJob

This packet is sent by the client to the server to indicate that the current job transmission will be stopped. Any portion of the file that was received will be deleted.

Required Priveleges

privSendJob

Request Parameters

queueID	identity of queue [long]
deviceID	identity of device [long]
jobID	identity of job [long]

Response Parameters

queueID	identity of queue [long]
deviceID	identity of device [long]
jobID	identity of job [long]

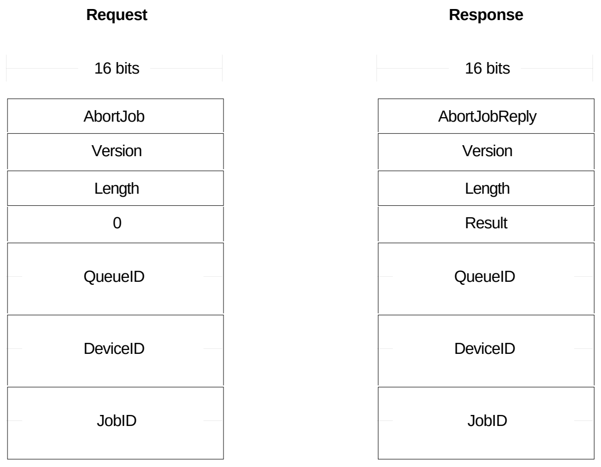
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation
psBadQueueID	no such queue
psBadDeviceID	no such device
psBadJobID	no such job

Algorithm

This protocol command stops the reception of the current job. The server is returned to the idle state and awaits the next command. All traces of the job are removed. The data channel is freed for both client and server.

Message Format



DeleteJob

The client sends this message when attempting to delete a pending (queued) job file.

Required Priveleges

privDeleteOwnJob | privDeleteGroupJob | privDeleteOtherJob

Request Parameters

queueID	identity of queue [long]
deviceID	identity of device [long]
jobID	identity of job [long]

Response Parameters

queueID	identity of queue [long]
deviceID	identity of device [long]
jobID	identity of job [long]

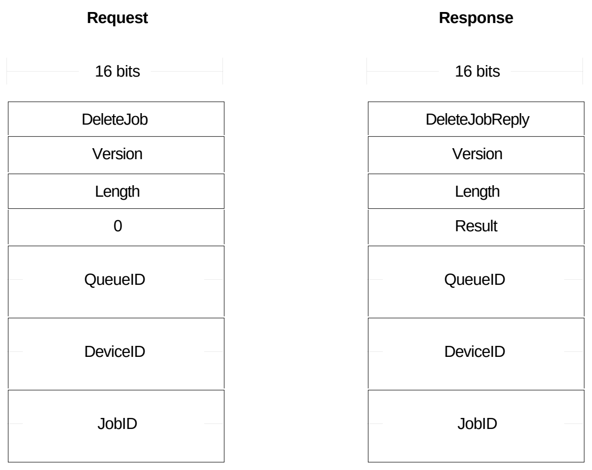
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation
psBadQueueID	no such queue
psBadDeviceID	no such device
psBadJobID	no such job

Algorithm

The server will remove the pending job from the queue. If the job is currently printing, the job is aborted as if an AbortJob call (see above) had been made. The job and all information pertaining to it is removed from the server system. It is an error to request information about a job that has been deleted.

Message Format



GetServerInfo

Request general information about the print server configuration.

Required Priveleges

privNone

Request Parameters

magicCookie	ServerInfo tag [long]

Response Parameters

magicCookie	ServerInfo tag [long]
flags	psPassword, psMagicCookies [unsigned short]
versionCount	number of APSP versions [short]
versions	APSP version(s) server supports [array of short]
uamCount	number of UAMs [short]
uams	UAM(s) supported by server [array of long]
serverName	server name [IntlStr31]

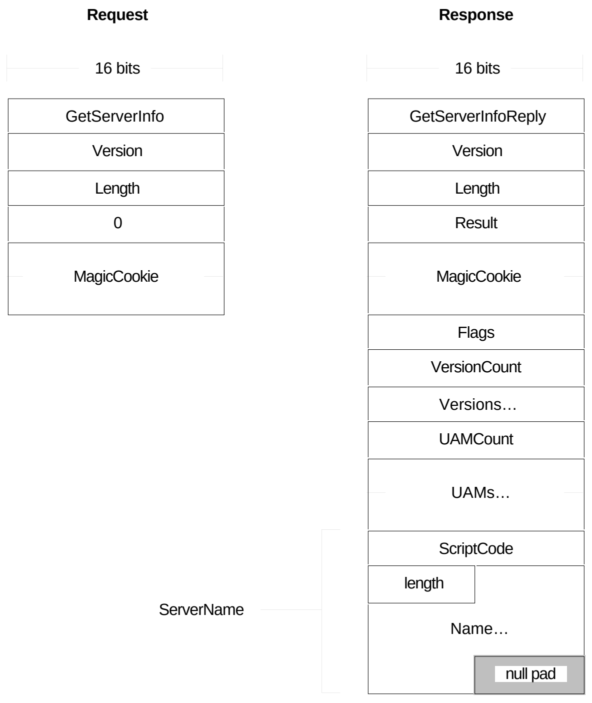
Result Codes

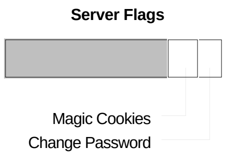
psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoInfo	server info not available
psNoChange	data has not changed since last request

Algorithm

This call returns information about the server in the form of an information block. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request. The psNoInfo result code should be returned if the server cannot complete the request because one or more of the return parameters are unavailable. Non-Macintosh servers should use zero (0) for the script code of the server name. The server flags bitfield describes whether the server supports the ChangePassword command, and whether the server supports magic cookies. A server implementation is not required to support magic cookies. All other flags are reserved, and should be set to zero (0).

Message Format





GetQueueList

This returns the list of queues for the server.

Required Priveleges

privNone

Request Parameters

magicCookie	QueueList tag [long]

Response Parameters

magicCookie	QueueList tag [long]
idCount	number of queue IDs [short]
queueIDs	queue IDs [array of long]

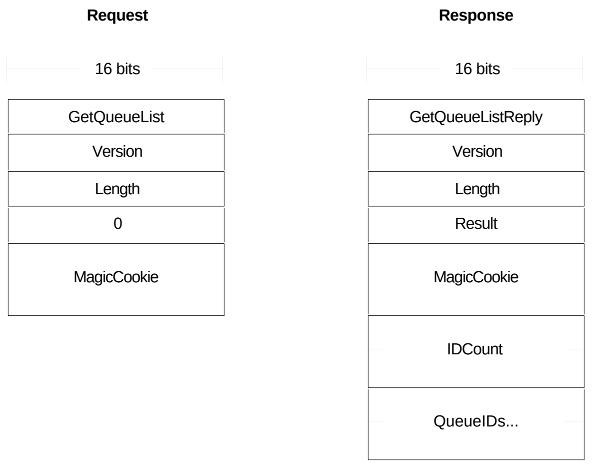
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request

Algorithm

The client requests the list of queueIDs for configured queues. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request.

Message Format



GetQueueInfo

This returns information about queues for the server.

This command is not supported in Version 1.0 of the protocol; it is provided merely for completeness' sake. See Future Directions below for more information.

Required Priveleges

privNone

Request Parameters

queueID	queueID to return info for [long]
magicCookie	QueueInfo tag [long]

Response Parameters

queueID	queueID to return info for [long]
magicCookie	QueueInfo tag [long]
queueInfo	queue information [TBD]

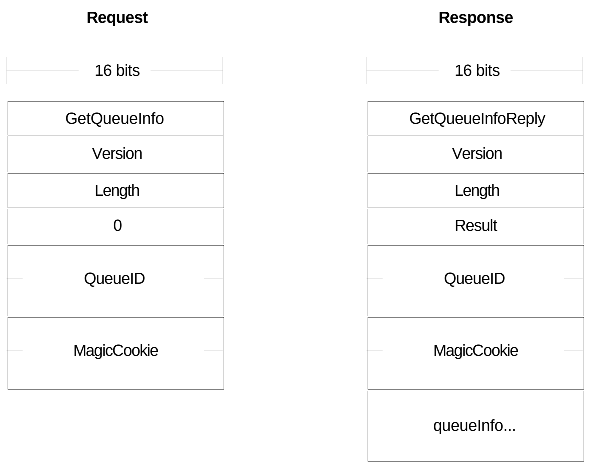
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request
psNoInfo	no queue info available
psBadQueueID	no such queue

Algorithm

Return a block of information describing the queue.

Message Format



SetQueueInfo

This changes information about queues for the server.

This command is not suypported in Version 1.0 of the protocol; it is provided merely for completeness' sake. See Future Directions below for more information.

Required Priveleges

privSetQueueInfo

Request Parameters

queueID	queueID to return info for [long]
magicCookie	QueueInfo tag [long]
queueInfo	queue information [TBD]

Response Parameters

queueID	queueID to return info for [long]
magicCookie	QueueInfo tag [long]

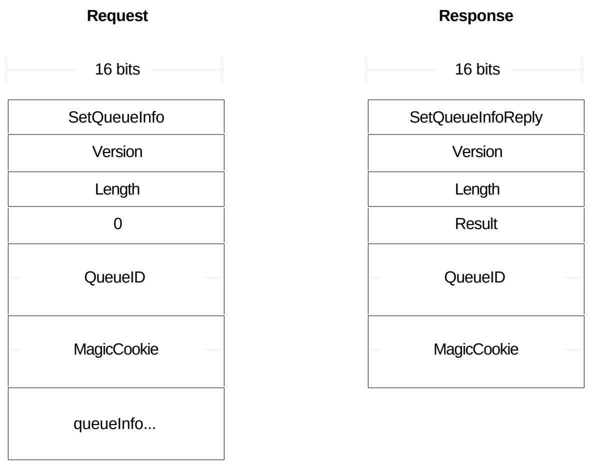
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psBadQueueID	no such queue
psNoPriv	no privelege for attempted operation

Algorithm

Return a block of information describing the queue.

Message Format



GetQueueState

This returns the list of queues for the server.

Required Priveleges

privNone

Request Parameters

queueID	queueID to return state for [long]
magicCookie	QueueState tag [long]

Response Parameters

queueID	queueID to return state for [long]
magicCookie	QueueState tag [long]
queueState	queue state bitfield [unsigned long]

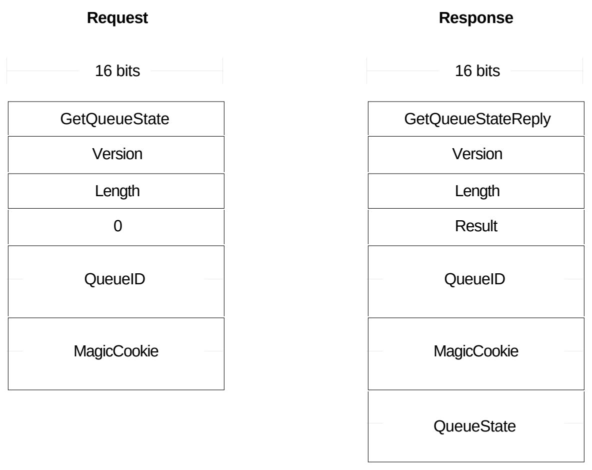
Result Codes

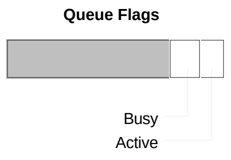
psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request
psBadQueueID	no such queue

Algorithm

The client requests the state for the specified queue. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request. The state information is returned as a bitfield, shown below as Queue Flags. All other bits are reserved and should be set to zero (0). The active bit means the queue is operational; the busy bit means a job is currently printing on one (or more) of the queue's devices.

Message Format




Queue Flags
Busy
Active

SetQueueState

This returns the list of queues for the server.

Required Priveleges

privSetQueueState

Request Parameters

queueID	queueID to return info for [long]
magicCookie	QueueState tag [long]
queueState	queue state bitfield [unsigned long]

Response Parameters

queueID	queueID to return state for [long]
magicCookie	QueueState tag [long]

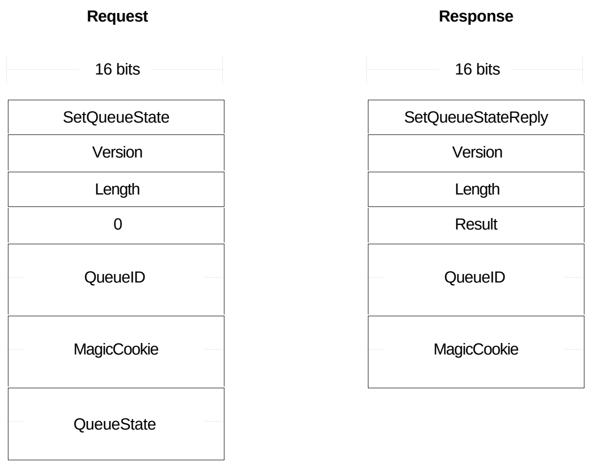
Result Codes

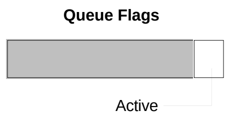
psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation
psBadQueueID	no such queue

Algorithm

The client wants to change a queue's state. A queue can be toggled between active and inactive. All other bits are reserved, and should be set to zero (0).

Message Format





GetDeviceList

This returns the list of dev ices for the queue.

Required Priveleges

privNone

Request Parameters

queueID	queueID for devices [long]
magicCookie	DeviceList tag [long]

Response Parameters

queueID	queueID for devices [long]
magicCookie	DeviceList tag [long]
idCount	number of device IDs [short]
deviceDs	device IDs [array of long]

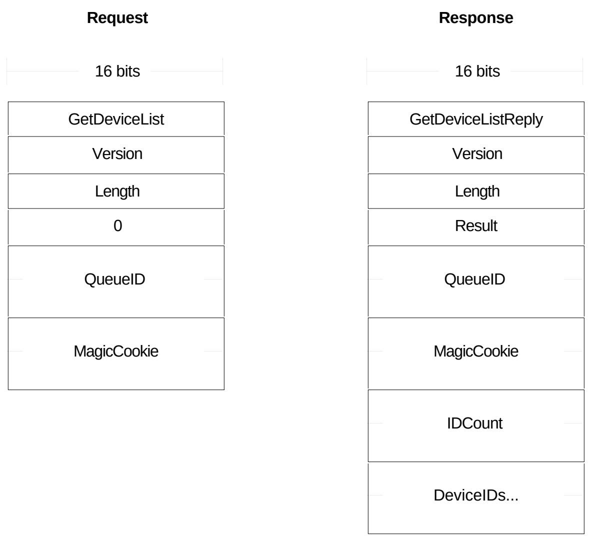
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request
psBadQueueID	no such queue

Algorithm

The client requests the list of deviceIDs for configured for the specified queue. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request.

Message Format



GetDeviceInfoList

This returns the list of info types for the device.

Required Priveleges

privConnect

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	DeviceInfoList tag [long]

Response Parameters

queueID	queueID for devices [long]
deviceID	deviceID for info [long]
magicCookie	DeviceList tag [long]
infoCount	number of infoTuples [short]
infoTuples	info tuples [array of long]

Result Codes

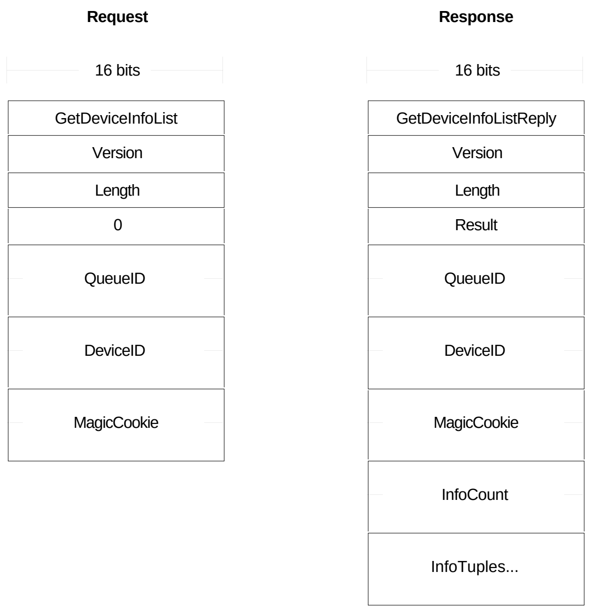
psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request
psBadQueueID	no such queue
psBadDeviceD	no such device
psNoPriv	no privelege for attempted operation

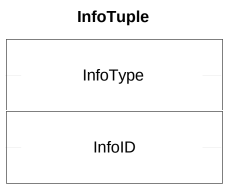
Algorithm

The client requests the list of info types for the specified device. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request.

InfoTuples are defined as an InfoType (unsigned long) followed by an InfoID (long). InfoTuples are essentially Macintosh resource type and id's. The high 16 bits of the InfoID is reserved for Macintosh resource attributes, and should be set to zero (0) unless otherwise indicated.

Message Format





GetDeviceInfo

This returns one piece of device info.

Required Priveleges

privConnect

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
infoType	DeviceInfo type [long]
infoID	DeviceInfo ID [long]
magicCookie	DeviceInfo tag [long]

Response Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	DeviceInfo tag [long]
devInfo	DeviceInfo [PSPParameter]

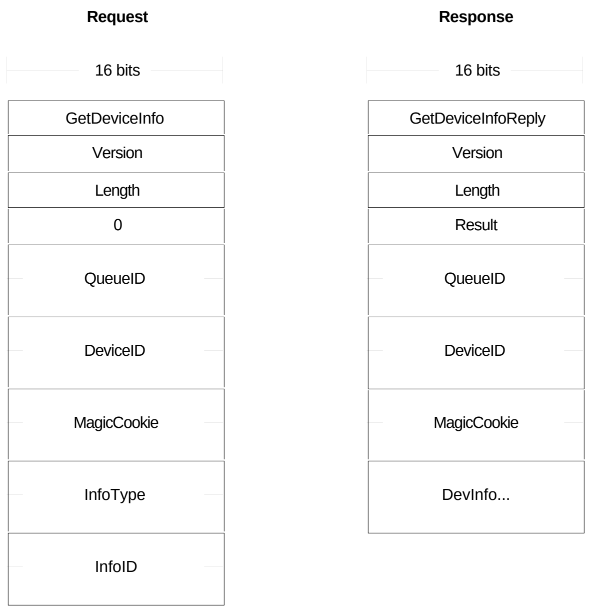
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request
psBadQueueID	no such queue
psBadDeviceD	no such device
psNoInfo	no info of specified type and ID
psNoPriv	no privelege for attempted operation

Algorithm

The client requests one of the DeviceInfo ID and tags from GetDeviceInfoList. This information is then returned to the client.

Message Format



GetDeviceStatus

This returns the list of queues for the server.

Required Priveleges

privGetOwnJobInfo | privGetGroupJobInfo | privGetOtherJobInfo

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	DeviceStatus tag [long]

Response Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	DeviceStatus tag [long]
jobID	currently printing job [long]
userCount	# of users connected to server [long]
currentPage	currently printing page [long]
statusSize	size of driver status [long]
status	driver status [array of char]

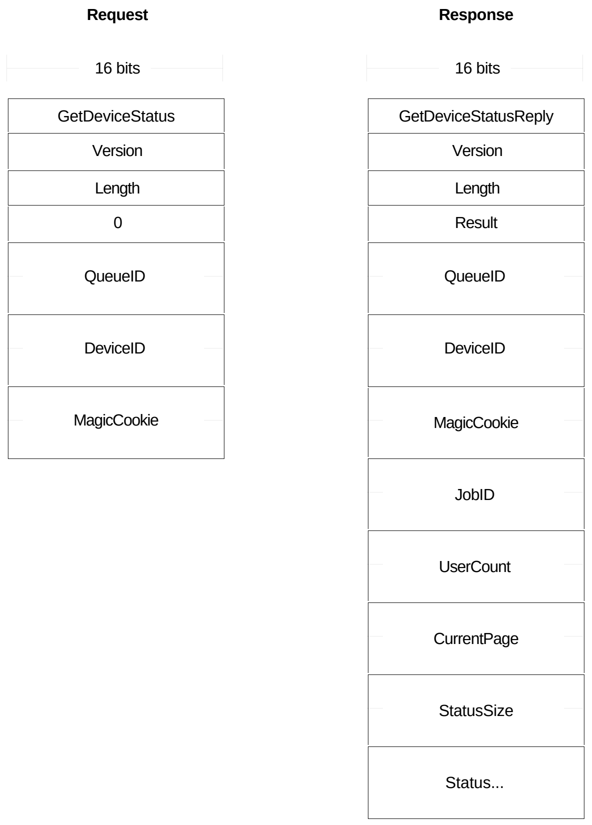
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoChange	data has not changed since last request
psBadQueueID	no such queue
psBadDeviceD	no such device
psNoPriv	no privelege for attempted operation

Algorithm

The client requests the list of queueIDs for configured and operational queues. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request.

Message Format



SetDeviceStatus

This returns the list of queues for the server.

Required Priveleges

privSetOwnJobInfo | privSetGroupJobInfo | privSetOtherJobInfo

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	DeviceStatus tag [long]
jobID	currently printing job [long]
statusSize	size of driver status [long]
status	driver status [array of char]

Response Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	DeviceStatus tag [long]

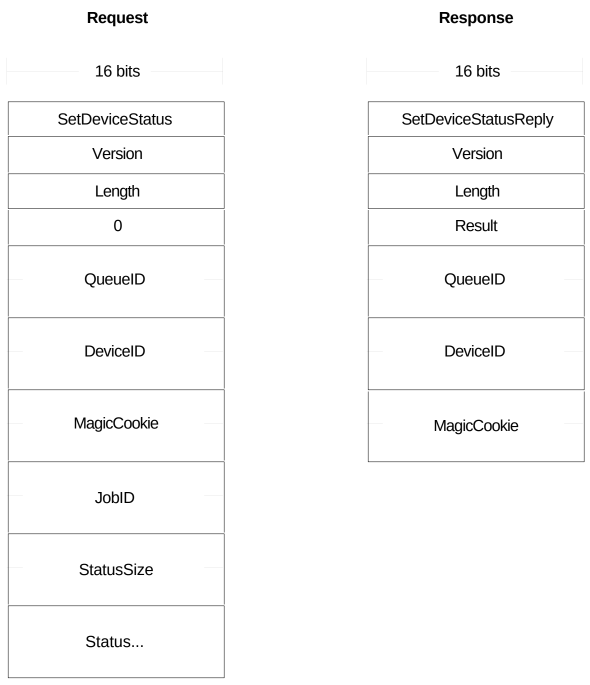
Result Codes

psNoErr	request completed normally
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psBadQueueID	no such queue
psBadDeviceD	no such device
psBadJobD	job or driver status has changed since GetDeviceStatus
psNoPriv	no privelege for attempted operation

Algorithm

The client requests the list of queueIDs for configured and operational queues. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request.

Message Format


Request
16 bits
SetDeviceStatus
Version
Length
0
QueueID
DeviceID
MagicCookie
JobID
StatusSize
Status...
Response
16 bits
SetDeviceStatusReply
Version
Length
Result
QueueID
DeviceID
MagicCookie

GetJobList

This returns the list of jobs in the specified queue.

Required Priveleges

privConnect

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	JobList tag [long]

Response Parameters

queueID	queueID for device [long]
deviceID	deviceID for info [long]
magicCookie	JobList tag [long]
idCount	number of job IDs [short]
jobIDs	job IDs [array of long]

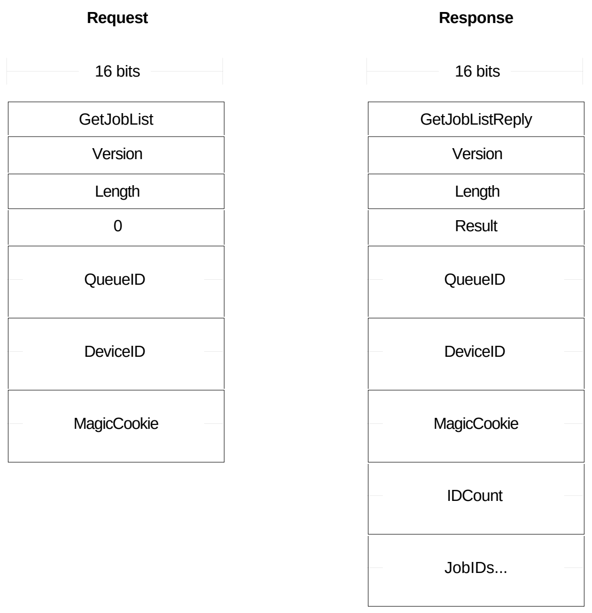
Result Codes

psNoErr	password was changed
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation
psNoChange	data has not changed since last request
psBadQueueID	no such queue
psBadDeviceID	no such device

Algorithm

The client requests the list of jobIDs for jobs pending on the specified queue. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request. The noInfo result code should be returned if no jobs are pending in the queue.

Message Format


Request
Response
16 bits
16 bits
GetJobList
Version
Length
0
QueueID
DeviceID
MagicCookie
GetJobListReply
Version
Length
Result
QueueID
DeviceID
MagicCookie
IDCount
JobIDs...

GetJobInfo

This requests information about a specific job.

Required Priveleges

privGetOwnJobInfo | privGetGroupJobInfo | privGetOtherJobInfo

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for job [long]
jobID	jobID for info [long]
magicCookie	JobInfo tag [long]

Response Parameters

queueID	queueID for device [long]
deviceID	deviceID for job [long]
jobID	jobID for info [long]
magicCookie	JobInfo tag [long]
priority	job priority [long]
timeToPrint	job scheduled time to print [long]
jobTimeout	job timeout in ticks [long]
jobAlert	job user interaction [long]
numPages	job size [long]
jobType	job type [unsigned long]
appName	application name [Str31]
docName	document name [Str31]
userName	user name [Str31]

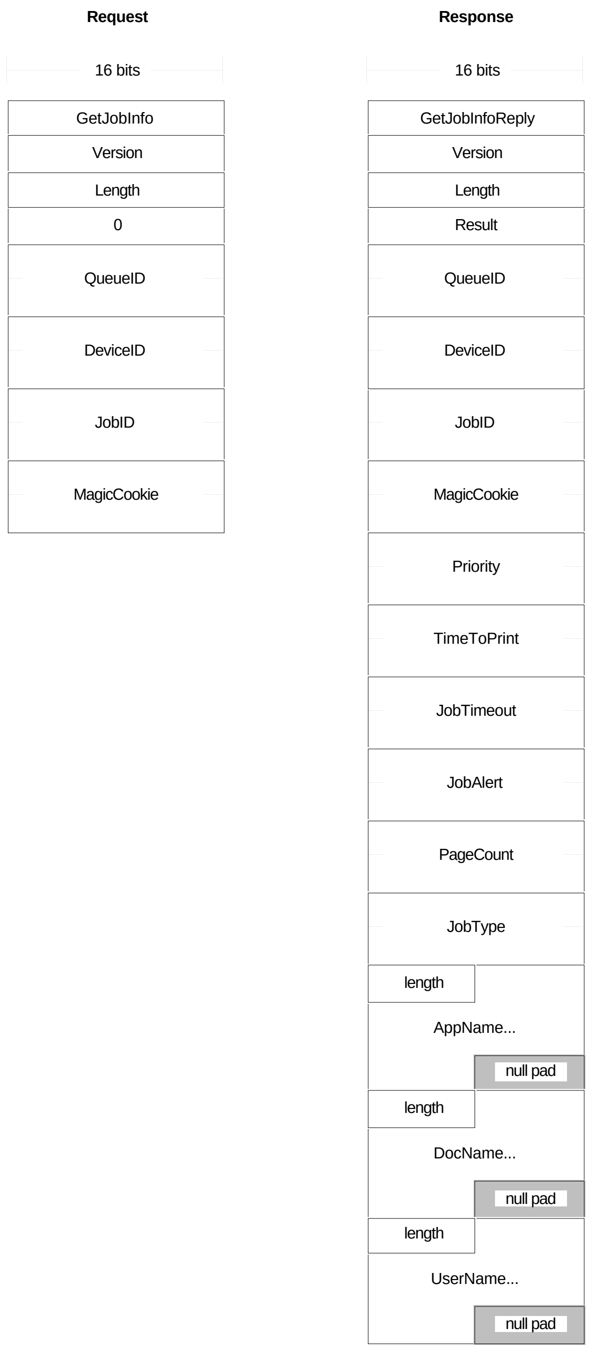
Result Codes

psNoErr	password was changed
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation
psNoInfo	no info available
psNoChange	data has not changed since last request
psBadQueueID	no such queue
psBadDeviceID	no such device
psBadJobID	no such job

Algorithm

This protocol command allows a client to request information on a single job. If the server supports magic cookies, the server may elect to send a psNoChange result code, indicating that the data has not changed since the last request.

Message Format



SetJobInfo

This sets information about a specific job.

Required Priveleges

privSetOwnJobInfo | privSetGroupJobInfo | privSetOtherJobInfo

Request Parameters

queueID	queueID for device [long]
deviceID	deviceID for job [long]
jobID	jobID for info [long]
magicCookie	JobInfo tag [long]
priority	job priority [long]
timeToPrint	job scheduled time to print [long]
jobTimeout	job timeout in ticks [long]
jobAlert	job user interaction [long]

Response Parameters

queueID	queueID for device [long]
deviceID	deviceID for job [long]
jobID	jobID for info [long]
magicCookie	JobInfo tag [long]

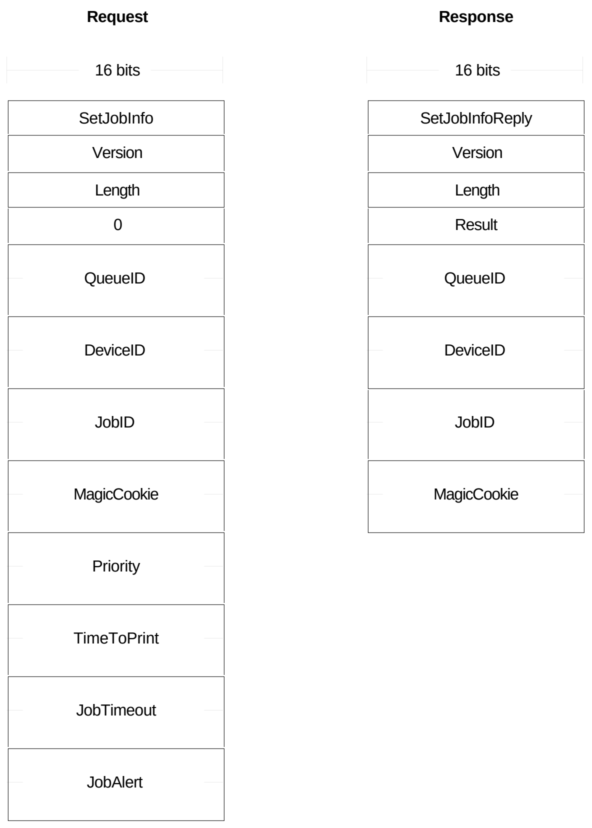
Result Codes

psNoErr	password was changed
psShutDown	server is shutting down
psSnafu	server exploded with happiness; please try again
psServerBusy	server cannot service request now; try again later
psNoPriv	no privelege for attempted operation
psBadQueueID	no such queue
psBadDeviceID	no such device
psBadJobID	no such job

Algorithm

This protocol command allows a client to set information on a single job. The badJobInfo result code should be used if the server detects that the jobInfo parameter contains invalid data, or if the server is unable to effect the requested change.

Message Format


Request
16 bits
SetJobInfo
Version
Length
0
QueueID
DeviceID
JobID
MagicCookie
Priority
TimeToPrint
JobTimeout
JobAlert
Response
16 bits
SetJobInfoReply
Version
Length
Result
QueueID
DeviceID
JobID
MagicCookie