GNUspool Release 1
API Reference Manual

This manual is for GNUspool (API Reference Manual).

Copyright 2008 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the GNUSpool reference manual in the section entitled ``GNU Free Documentation License''.

Table of Contents

1 Introduction to GNUspool API

The **GNUspool** API enables a C or C++ programmer to access **GNUspool** facilities directly from within an application. The application may be on a Unix host or on a Windows workstation.

Communication takes place using a TCP connection between the API running on a Windows or Unix machine and the server process **xtnetserv** running on the Unix host in question. The same application may safely make several simultaneous conversations with the same or different host.

The user may submit, change, delete and alter the state of jobs or printers to which he or she has access, and may receive notification about changes which may require attention. In addition, the user access control parameters may be viewed and if permitted, changed.

2 Installation and access to API

The API is provided as two files, a header file `gspool.h` and a library file.

The header file should be copied to a suitable location for ready access.

```
#include <gspool.h>
```

The library file is supplied in the form `libgnuspool.a` or as a shared `library` `libgnuspool.so` on Unix systems. This may need be copied or linked to `/lib` or `/usr/lib` so that it may be linked with the option `-lgnuspool` when the program is compiled. On some systems you may have to include a socket handling library as well. The shared library is usually placed in `/usr/local/lib`.

On Windows systems the library is supplied as gnuspool.dll. Again we suggest that it be placed in the default search path.

3 The API file descriptor

Each routine in the API uses a file descriptor to identify the instance in progress. This is an integer value, and is returned by a successful call to the gspool_open or gspool_login routine. All other routines, apart from job string manipulation routines, take this value as a first parameter. As mentioned before, more than one session may be in progress at once with different gspool_open parameters.

Each session with the API should be commenced with a call to gspool_open or gspool_login and terminated with a call to gspool_close.

3.1 Error return codes

Nearly all the routines return an integer response code. This is usually zero to indicate success (except for gspool_open which returns a positive or zero file descriptor). The error codes are described below.

Those routines which return a pointer to a FILE structure return NULL on error and put the error code in gspool_dataerror.

Code	Name	Meaning
0	GSPOOL_OK	No error, successful completion
-1	GSPOOL_INVALID_FD	The file descriptor argument is invalid.
-2	GSPOOL_NOMEM	Run out of memory allocation within API library.
-3	GSPOOL_INVALID_HOSTNAME	Invalid host name in gspool_open
-4	GSPOOL_INVALID_SERVICE	Invalid service name in gspool_open
-5	GSPOOL_NODEFAULT_SERVICE	Default service relied upon and no default API service set up in services file
-6	GSPOOL_NOSOCKET	Cannot create socket
-7	GSPOOL_NOBIND	Cannot bind address to socket
-8	GSPOOL_NOCONNECT	Connection refused by server
-9	GSPOOL_BADREAD	Read error on socket
-10	GSPOOL_BADWRITE	Write error on socket
-11	GSPOOL_CHILDPROC	Cannot fork to make child process
-23	GSPOOL_UNKNOWN_USER	User invoking API is unknown on server
-24	GSPOOL_ZERO_CLASS	Class code is effectively zero
-25	GSPOOL_BAD_PRIORITY	Invalid priority (outside permitted range)
-26	GSPOOL_BAD_COPIES	Invalid number of copies (above limit)
-27	GSPOOL_BAD_FORM	Invalid form type (user is restricted)
-28	GSPOOL_NOMEM_QF	No memory for queue file on server
-29	GSPOOL_BAD_PF	Cannot open page file
-30	GSPOOL_NOMEM_PF	No memory for page file on server
-31	GSPOOL_CC_PAGEFILE	Cannot create page file
-32	GSPOOL_FILE_FULL	Server file system is full
-33	GSPOOL_QFULL	Server message queue full

Code	Name	Meaning
-34	GSPOOL_EMPTYFILE	Job file is empty
-35	GSPOOL_BAD_PTR	Invalid printer name (user restricted)
-36	GSPOOL_WARN_LIMIT	Job exceeds limit, truncated
-37	GSPOOL_PAST_LIMIT	Job exceeds limit, not queued
-38	GSPOOL_NO_PASSWD	Password required and not given
-39	GSPOOL_PASSWD_INVALID	Invalid password
-40	GSPOOL_UNKNOWN_COMMAND	Unknown API operation (error in library)
-41	GSPOOL_SEQUENCE	Sequence error, operation(s) since last read
-42	GSPOOL_UNKNOWN_JOB	Job not found
-43	GSPOOL_UNKNOWN_PTR	Printer not found
-44	GSPOOL_NOPERM	No privilege for operation
-45	GSPOOL_NOTPRINTED	Job has not been printed
-46	GSPOOL_PTR_NOTRUNNING	Printer not running
-47	GSPOOL_PTR_RUNNING	Printer is running
-48	GSPOOL_PTR_NULL	Null printer name
-49	GSPOOL_PTR_CDEV	No permission to change device
-50	GSPOOL_INVALIDSLOT	Invalid slot number

4 Slot numbers

Each job or printer is identified to **GNUspool** by means of two numbers:

1. The host or network identifier. This is a long corresponding to the internet address in network byte order. The host identifier is given the type `netid_t`.

2. The shared memory offset, or slot number. This is the offset in shared memory on the relevant host of the job or printer and stays constant during the lifetime of the job or printer. The type for this is `slotno_t`.

These two quantities uniquely identify any job or printer.

It might be worth noting that there are two slot numbers relating to a remote job or printer.

1. The slot number of the record of the job or printer held in local shared memory. This is the slot number which will in all cases be manipulated directly by the API.

2. The slot number of the job on the owning host. This is in fact available in the job structures as the field `apispq_rslot` and in the printer structure as the field `apispp_rslot`.

These fields usually have the same value as the slot number in local memory for local jobs or printers, but this should not be relied upon.

5 Sequence numbers

These quantities are not available directly, but are held to determine how out-of-date the user's record of jobs or printers may be.

Every time you read a job or printer record, the sequence number of the job or printer list is checked, and if out-of-date, you will receive the error GSPOOL_SEQUENCE. This is not so much of an error as a warning. If you re-read the job or printer required, then you will not receive this error.

If you want to bypass this, you can access the job or printer without worrying about the sequence using the flag GSPOOL_FLAG_IGNORESEQ, however you might receive an error about unknown job or printer if the job or printer has disappeared.

6 API Functions

The following sub-sections describe the **GNUspool** API C routines including each function's purpose, syntax, parameters and possible return values.

The function descriptions also contain additional information that illustrate how the function can be used to carry out tasks.

In some cases there are slight differences between the Unix and Windows variants, these are noted where appropriate.

6.1 Sign-on and off

6.1.1 gspool_open (Unix and GNU/Linux versions)

```
int gspool_open(const char *host,
            const char *serv,
            const classcode_t cl)
```

The gspool_open routine is used to commence a session with the API on Unix or GNU/Linux hosts, where the user name is obtained from the effective user id invoking the routine.

Host is the name of the host to be connected to. Serv is the name of the TCP/UDP service to be used to connect to **GNUspool**. If this is set to NULL, the default service will be used (but this must be set up correctly).

Cl is the classcode to be used for access to all printer and job operations. If 0 is used the user's default class code will be used. Note that the class code bits are assigned starting at the least significant bit, thus:

```
1 = A
2 = B
4 = C
....
0x8000 = P
0x10000 = a
0x20000 = b
0x40000 = c
....
0x80000000 = p
```

The function returns an integer descriptor greater than or equal to 0 on success. This descriptor should be used in all subsequent operations with the API.

If an error occurs, one of the (negative) error codes will be returned as listed in section 3.1.

Any number of API sessions, to the same host or to various hosts, may be in progress simultaneously, subject to the restrictions on the number of simultaneous open files per process which the operating system allows.

Each connection should also be terminated by a call to gspool_close or exit from the calling program.

An example to open a connection to host "xisl":

```
int fd, ret;
fd = gspool_open("xisl", (char *) 0, 0);
if (fd < 0) {
    /* error handling */
        ...
}

/* process connection */

gspool_close(fd);
```

6.1.2 gspool_open (Windows version)

```
int gspool_open(const char *host,
            const char *serv,
            const char *user,
            const classcode_t cl)
```

The gspool_open routine is used to commence a session with the API. This second form is that used on Windows hosts, where the Windows machine has a static IP address and no password is required. See the next section for the form used on Windows hosts with IP addresses allocated via DHCP.

Host is the name of the host to be connected to. Serv is the name of the TCP/UDP service to be used to connect to **GNUspool**. If this is set to NULL, the default service will be used (but this must be set up correctly).

User is the Unix user name to be used.

Cl is the classcode to be used for access to all printer and job operations. If 0 is used the user's default class code will be used. Note that the class code bits are assigned starting at the least significant bit, thus:

```
1 = A
2 = B
4 = C
....
0x8000 = P
0x10000 = a
0x20000 = b
0x40000 = c
....
0x80000000 = p
```

The function returns an integer descriptor greater than or equal to 0 on success. This descriptor should be used in all subsequent operations with the API.

If an error occurs, one of the (negative) error codes will be returned as listed in section 3.1.

Each connection should also be terminated by a call to gspool_close or exit from the calling program.

6.1.3 gspool_login (Windows version)

```
int gspool_login(const char *host,
                 const char *serv,
                 const char *user,
                 char *pass,
                 const classcode_t cl)
```

This routine is used on Windows hosts with IP addresses allocated via DHCP and/or where a password is required.

Host is the name of the host to be connected to. Serv is the name of the TCP/UDP service to be used to connect to **GNUspool**. If this is set to NULL, the default service will be used (but this must be set up correctly).

User is the Unix user name to be used on Windows hosts and pass is the password. Note that this is not const, and the passed field will be deliberately overwritten as soon as possible within gspool_login. Also note that the password may not be the Unix password, it may be an interface password set up by xipasswd.

Cl is the classcode to be used for access to all printer and job operations. If 0 is used the user's default class code will be used. Note that the class code bits are assigned starting at the least significant bit, thus:

```
1 = A
2 = B
4 = C
....
0x8000 = P
0x10000 = a
0x20000 = b
0x40000 = c
....
0x80000000 = p
```

The function returns an integer descriptor greater than or equal to 0 on success. This descriptor should be used in all subsequent operations with the API.

If an error occurs, one of the (negative) error codes will be returned as listed in section 3.1.

Each connection should also be terminated by a call to gspool_close or exit from the calling program.

6.1.4 gspool_close

```
int gspool_close(const int fd)
```

The gspool_close function is used to close a connection to GNUspool. Fd is a file descriptor previously returned by a successful call to gspool_open or gspool_login.

gspool_close returns 0 if successful or GSPOOL_INVALID_FD (Invalid File descriptor, a constant defined in gspool.h) if the passed file descriptor was not valid, perhaps because it was never opened successfully.

6.2 Job Operations

6.2.1 gspool_joblist

```
int gspool_joblist(const int fd,
                   const unsigned flags,
                   int *numjobs,
                   slotno_t **slots)
```

The gspool_joblist function is used to obtain a list of jobs.

Fd is a file descriptor previously returned by a successful call to gspool_open

Flags is zero, or a logical OR of one or more of the following values

GSPOOL_FLAG_LOCALONLY	Ignore remote printers/hosts, i.e. not local to the server, not the client.
GSPOOL_FLAG_USERONLY	Ignore other users jobs

Numjobs is a pointer to an integer value which, on successful completion, will contain the number of job slots returned.

Slots is a pointer to to an array of slot numbers. These slot numbers can be used to access individual jobs. The memory used by this vector is owned by the API, therefore no attempt should be made by the user to free it. This contrasts, for example, with X library routines. Also note that certain other calls to the API, notably gspool_ptrlist, with the same value of fd, may reuse the space, so the contents should be copied if required before other API calls are made.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to list all jobs:

```
int fd, ret, nj, i;
slotno_t *slots;
fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
     ...
}
ret = gspool_joblist(fd, 0, &nj, &slots);
if (ret < 0) { /* error handling */
     ...
}
for (i = 0; i < nj; i++) {
     slotno_t this_slot = slots[i];
     /* process this_slot */
     ...
}
gspool_close(fd);
```

6.2.2 gspool_jobread

```
int gspool_jobread(const int fd,
                   const unsigned flags,
                   const slotno_t slot,
                   struct apispq *jobd)
```

The gspool_jobread function is used to retrieve the details of a job from a given slot number.

Fd is a file descriptor previously returned by gspool_open

Flags is zero, or a logical OR of one or more of the following values

GSPOOL_FLAG_LOCALONLY	Ignore remote printers/hosts, i.e. not local to the server, not the client.
GSPOOL_FLAG_USERONLY	Ignore other users jobs
GSPOOL_FLAG_IGNORESEQ	Ignore changes since the list was last read

Slot is the slot number corresponding to the job as returned by gspool_joblist or gspool_jobfindslot.

Jobd is a descriptor, which on return will contain the details of the job in a struct apispq as defined in gspool.h and containing the following elements:

Type	Field	Description
jobno_t	apispq_job	Job number
netid_t	apispq_netid	Host address (network byte order)
netid_t	apispq_orighost	Originating host address
slotno_t	apispq_rslot	Slot number on owning machine
time_t	apispq_time	Time job was submitted
time_t	apispq_starttime	Time job was started (if applicable)
time_t	apispq_hold	Time job held to, 0 if not held
unsigned short	apispq_nptimeout	Time after to delete job if not printed (hours)
unsigned short	apispq_ptimeout	Time after to delete job if printed (hours)
unsigned short	apispq_extrn	External job type index
unsigned short	apispq_pglim	Job size limit applies
long	apispq_size	Size of job in bytes
long	apispq_posn	Offset reached if currently being printed
long	apispq_pagec	Currently-reached page if being printed
char[]	apispq_uname	User name of job owner
char[]	apispq_puname	User name of posting user
unsigned char	apispq_cps	Copies
unsigned char	apispq_pri	Priority
classcode_t	apispq_class	Class code bits 1=A 2=B 4=C etc
unsigned short	apispq_jflags	Job flags
unsigned char	apispq_dflags	Despooler flags

slotno_t	apispq_pslot	Printer slot assigned to if printing
unsigned long	apispq_start	Start page 0=first page
unsigned long	apispq_end	End page
unsigned long	apispq_npages	Number of pages
unsigned long	apispq_haltat	"Halted at" page
char []	apispq_file	Job title
char []	apispq_form	Job form type
char []	apispq_ptr	Printer pattern assigned to job
char []	apispq_flags	Post-processing flags

The following bits are set in the `apispq_jflags` field to indicate job parameters:

Bit (#define)	Meaning
APISPQ_NOH	Suppress header
APISPQ_WRT	Write result
APISPQ_MAIL	Mail result
APISPQ_RETN	Retain on queue after printing
APISPQ_ODDP	Suppress odd pages
APISPQ_EVENP	Suppress even pages
APISPQ_REVOE	Invert APISPQ_ODDP and API_EVENP after printing
APISPQ_MATTN	Mail attention
APISPQ_WATTN	Write attention
APISPQ_LOCALONLY	Handle job on local machine only
APISPQ_CLIENTJOB	Job originated with windows client
APISPQ_ROAMUSER	Job originated with DHCP windows client

The `apispq_dflags` field contains the following bits:

Bit (#define)	Description
APISPQ_PQ	Job being printed
APISPQ_PRINTED	Job has been printed
APISPQ_STARTED	Job has been started
APISPQ_PAGEFILE	Job has a page file
APISPQ_ERRLIMIT	Error if size limit exceeded
APISPQ_PGLIMIT	Size limit in pages not KB

Note that the field `apispq_pglim` and the field bits `APISPQ_ERRLIMIT` and `APISPQ_PGLIMIT` will always be zero when read, but the description is included for completeness. The fields are only used when creating jobs.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to read the names of all jobs

```
int fd, ret, nj, i;
struct apispq job;
slotno_t *slots;
```

```
fd = gspool_open("myhost", (char *)0, 0);
if (fd < 0) { /* error handling */
    ...
}

ret = gspool_joblist(fd, 0, &nj, &slots);
if (ret < 0) { /* error handling */
    ...
}

for (i = 0; i < nj, i++) {
    ret = gspool_jobread(fd, 0, slots[i], &job);
    if (ret < 0) { /* error handling */
        ...
    }
    printf("%s\n", job.apispq_file);
}

gspool_close(fd);
```

6.2.3 gspool_jobfind

```
int gspool_jobfind(const int fd,
                const unsigned flags,
                const jobno_t jobnum,
                const netid_t nid,
                slotno_t *slot,
                struct apispq *jobd)

int gspool_jobfindslot(const int fd,
                    const unsigned flags,
                    const jobno_t jobnum,
                    const netid_t nid,
                    slotno_t *slot)
```

The gspool_jobfind and gspool_jobfindslot functions may be used to find a job from a given job number rather than by the slot number. gspool_jobfind retrieves the job descriptor, gspool_jobfindslot just retrieves the slot number.

Fd is a file descriptor previously returned by gspool_open

Flags is zero, or a logical OR of one or more of the following values

GSPOOL_FLAG_LOCALONLY	Ignore remote printers/hosts, i.e. not local to the server, not the client.
GSPOOL_FLAG_USERONLY	Ignore other users jobs

Jobnum is the job number to be searched for.

Nid is the network-byte order IP address of the host of the machine whose job is to be searched for. This should be correct even if GSPOOL_FLAG_LOCALONLY is specified.

Slot is a pointer to a location in which the slot number of the job is placed if the search is successful. It may be NULL if this is not required (but this would be almost

pointless for gspool_jobfindslot).

Jobd is a descriptor containing the job descriptor as defined in gspool.h.

The fields in struct apispq are defined in the gspool_jobread documentation in section 6.2.2.

The functions return 0 if successful otherwise one of the error codes as listed in section 3.1.

6.2.4 gspool_jobdata (Unix and GNU/Linux)

```
FILE *gspool_jobdata(const int fd,
                     const unsigned flags,
                     const slotno_t slotno)
```

The function gspool_jobdata is used to retrieve the job file of a job.

Fd is a file descriptor previously returned by gspool_open.

Flags is zero, or GSPOOL_FLAG_IGNORESEQ to ignore changes since the job list was last read.

Slotno is the slot number corresponding to the job previously returned by functions such as gspool_joblist or gspool_jobfindslot.

The result is a FILE pointer which can be used with all standard I/O input functions such as fgets(3), getc(3) etc. At the end of the data fclose(3) must be called. For reasons of synchronisation the file should be read to the end before other operations are attempted.

If an error is detected, gspool_jobdata returns NULL and an error code is placed in the external variable gspool_dataerror. This will be one of the error codes listed in section 3.1.

An example to retrieve the data for a job:

```
int     fd, ret, ch;
slotno_t slot, *list;
FILE *inf;
fd = gspool_open("myhost", (char *) 0, 0);
if  (fd < 0) { /* error handling */
    ...
}
/* Select a job slot and assign this to "slot" */
    .......
inf = gspool_jobdata(fd, 0, slot);
if (!inf)  { /* handle errors */
    ...
}
while ((ch = getc(inf)) != EOF)
    putchar(ch);
fclose(inf);
gspool_close(fd);
```

6.2.5 gspool_jobdata (Windows version)

```
int gspool_jobdata(const int fd,
                   const int outfile,
                   int (*func)(int, void*, unsigned),
                   const unsigned flags,
                   const slotno_t slotno)
```

This format of the gspool_jobdata function is for use by Windows programs, as there is no acceptable equivalent of the pipe(2) construct.

The second argument outfile is (possibly) a file handle to the file from to which the job data is passed as the first argument to func.

The third argument func is a function with the same specifications as write, indeed it may very well be write. The main reason for doing it this way is that some versions of Windows do strange things if write is invoked from within a DLL.

Other aspects of the interface are similar to the Unix routine, apart from the routine returning zero for success and an error code for failure rather than a FILE* or NULL. For consistency with the Unix version, the external variable gspool_dataerror is also assigned any error code returned. This will be one of the error codes listed in section 3.1.

6.2.6 gspool_jobpbrk (Unix and GNU/Linux versions)

```
FILE *gspool_jobpbrk(const int fd,
                     const unsigned flags,
                     const slotno_t slotno)
```

The function gspool_jobpbrk is used to retrieve the page break offset file of a job.

Fd is a file descriptor previously returned by gspool_open. Flags is zero, or GSPOOL_FLAG_IGNORESEQ to changes since the job list was last read.

Slotno is the slot number corresponding to the job previously returned by functions such as gspool_joblist or gspool_jobfindslot.

The result is a FILE pointer which can be used with all standard I/O input functions such as fread(3), fgets(3), getc(3) etc. At the end of the data fclose(3) must be called. For reasons of synchronisation the file should be read to the end before other operations are attempted.

If an error is detected, gspool_jobpbrk returns NULL and an error code is placed in the external variable gspool_dataerror. This will be one of the error codes listed in section 3.1.

If there is no page offset file, probably because the delimiter is set to formfeed, then this isn't really an error, but an error report of GSPOOL_BAD_PF will be returned. You can tell whether there is a page file from the struct apispq job structure returned by gspool_jobread or gspool_jobfind. The field apispq_dflags has the bit designated by APISPQ_PAGEFILE set if there is a page file.

The data is returned in three parts.

struct apipages This is an instance of the following structure, defined in gspool.h, and described below.

delimiter string This is the delimiter string itself,

A vector of longs giving the offsets of the start of each page, including the first page, which is always zero, within the job data (as read by gspool_jobdata).

The struct apipages structure is as follows:

```
struct apipages {
    long delimnum;  /* Number of delimiters */
    long deliml;    /* Length of delimiter string */
    long lastpage;  /* Number of delimiters remaining on last page */
};
```

6.2.7 gspool_jobpbrk (Windows version)

```
int gspool_jobpbrk(const int fd,
                const int outfile,
                int (*func)(int, void*, unsigned),
                const unsigned flags,
                const slotno_t slotno)
```

This second format of the gspool_jobpbrk function is for use by Windows programs, as there is no acceptable equivalent of the pipe(2) construct.

The second argument outfile is (possibly) a file handle to the file from to which the job data is passed as the first argument to func.

The third argument func is a function with the same specifications as write, indeed it may very well be write. The main reason for doing it this way is that some versions of Windows do strange things if write is invoked from within a DLL.

Other aspects of the interface are similar to the Unix routine, apart from the routine returning zero for success and an error code for failure rather than a FILE* or NULL. For consistency with the Unix version, the external variable gspool_dataerror is also assigned any error code returned.

6.2.8 gspool_jobadd (Unix and GNU/Linux versions)

```
FILE *gspool_jobadd(const int fd,
                struct apispq *jobd,
                const char *delim,
                const unsigned deliml,
                const unsigned delimnum)

int gspool_jobres(const int fd,
                jobno_t *jobno)
```

The functions gspool_jobadd and gspool_jobres are used to add a job under Unix and GNU/Linux.

Fd is a file descriptor previously returned by gspool_open

Jobd is a pointer to a `struct apispq`, as defined in `gspool.h and in section 6.2.2` containing all the details of the job. The fields in `struct apispq` are defined in the documentation.

Note that we recommend that the whole structure be cleared to zeroes initially and then required fields added; this approach will cover any future extensions with additional fields which will behave as at present if zero.

Also note that from release 23 an additional field is provided in the structure. If this is non-zero, then the size of the job is limited. If the bit `APISPQ_PGLIMIT` in is zero, then the size is limited to the given number of kilobytes. If this bit is set, then the size is limited to the given number of pages. If a job exceeds the given limit, then its treatment depends upon the setting of the bit `APISPQ_ERRLIMIT` in . If this is zero, then the job is truncated to the given number of kilobytes or pages and still proceeds (although a warning code is returned by gspool_jobres). If it is set, then it is rejected altogether.

Delim is a pointer to a string containing the page delimiter string, or NULL if the user is content with the single formfeed character. Deliml is the length of the delimiter string `delim`. This is necessary because `delim` is not necessarily null-terminated.

Delimnum in the number of instances of the delimiter string/character to be counted to make up a page.

The result is either a standard I/O stream, which can be used as output for putc(3), fprintf(3), fwrite(3) etc, or NULL to indicate an error has been detected. The I/O stream connection should be closed, when complete, with fclose(3). Finally a call should be made to gspool_jobres.

For reasons of synchronisation you must call gspool_jobres immediately after fclose(3) even if you are not interested in the answer. Apart from that several calls to gspool_jobadd may be in progress at once to submit several jobs simultaneously.

gspool_jobres returns zero on successful completion (or `GSPOOL_WARN_LIMIT` if the job was truncated but still submitted). The parameter jobno is assigned the job number of the job created. This value is also assigned to the field `apispq_job` in the passed structure jobd to gspool_jobadd.

Note that you should not call gspool_jobres if gspool_jobadd returns NULL for error. Most errors are detected at the gspool_jobadd stage and before any data is passed across, but this should not in general be relied upon.

An example to add a job called readme from standard input:

```
int fd, ret, ch;
struct apispq outj;
jobno_t jn;
FILE *f;

fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}
```

```
/* It is safest to clear the structure first */
memset((void *) &outj, '\0', sizeof(outj));

/* set defaults */

outj.apispq_nptimeout = 24 * 7;
outj.apispq_ptimeout = 24;
outj.apispq_cps = 1;
outj.apispq_pri = 150;

/* The class code specified in gspool_open is not used here. However
the
    user's class code will be &ed with this unless the user has
    override class privilege. */

outj.apispq_class = 0xffffffff;

/* set a large page range to to ensure all pages are printed */

outj.apispq_end = 4000;

/* Only the form type is compulsory here. The others may
    be set to NULL */

strcpy(outj.apispq_file, "readme");
strcpy(outj.apispq_form, "a4");
strcpy(outj.apispq_ptr, "laser");

/* add the job with the default page delimiter */

f = gspool_outjadd(fd, &outj, (char *) 0, 1, 1);
if (!f)  { /* error handling error in gspool_dataerror */
    ...
}

/* now send the data */

while ((ch = getchar()) != EOF)
    putc(ch, f);
fclose(f);

ret = gspool_jobres(fd, &jn);
if (ret < 0) { /* error handling */
    ...
} else
    printf("success the job number is %ld\n", jn);
gspool_close(fd);
```

6.2.9 gspool_jobadd (Windows version)

```
int gspool_jobadd(const int fd,
                   const int infile,
                   int (*func)(int, void*, unsigned).
                   struct apispq *jobd,
                   const char *delim,
                   const unsigned deliml.
                   const unsigned delimnum)
```

This second format of the gspool_jobadd function is for use by Windows programs, as there is no acceptable equivalent of the pipe(2) construct.

The second argument infile is (possibly) a file handle to the file from which the job is created and is passed as the first argument to func.

The third argument func is a function with the same specifications as read, indeed it may very well be read. The main reason for doing it this way is that some versions of Windows do strange things if read is invoked from within a DLL.

Other aspects of the interface are similar to the Unix routine, apart from the routine returning zero for success and an error code for failure rather than a FILE* or NULL.

There is no gspool_jobres in the windows version, the job number is placed in the field apispq_job in the passed structure jobd to gspool_jobadd. For consistency with the Unix version, the external variable gspool_dataerror is also assigned any error code returned.

6.2.10 gspool_jobdel

```
int gspool_jobdel(const int fd,
                  const unsigned flags,
                  const slotno_t slot)
```

The gspool_jobdel function is used to delete a job, aborting it if it is currently printing.

Fd is a file descriptor previously returned by gspool_open

Flags is zero, or the logical OR of one or both of the following:

GSPOOL_FLAG_IGNORESEQ Ignore changes since the list was last read
GSPOOL_FLAG_FORCE   Ignore "not printed" flag

Slot is the slot number corresponding to the job as previously returned by gspool_joblist or gspool_jobfindslot.

If the job has not been printed, and flags does not contain GSPOOL_FLAG_FORCE, then the job will not be deleted, but the error GSPOOL_NOT_PRINTED will be reported. You can tell whether the job has been printed from the struct apispq job structure returned by gspool_jobread or gspool_jobfind. The field apispq_dflags has the bit designated by APISPQ_PRINTED set if it has been printed.

The function returns 0 if successful otherwise one of the error codes listed in section 3.1.

An example to delete all jobs:

```
int fd, ret, nj, i;
slotno_t *slots;
fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}
ret = gspool_joblist(fd, 0, &nj, &slots);
if (ret < 0) { /* error handling */
    ...
}
for (i = 0; i < nj; i++) {
    ret = gspool_jobdel(fd, GSPOOL_FLAG_FORCE, slots[i]);
    if (ret < 0) { /* error handling */
        ...
    }
}
gspool_close(fd);
```

6.2.11 gspool_jobupd

```
int gspool_jobupd(const int fd,
                  const unsigned flags,
                  const slotno_t slot,
                  struct apispq * jobd)
```

The gspool_jobupd function is used to update the details of a job.

Fd is a file descriptor previously returned by gspool_open

Flags is zero, or GSPOOL_FLAG_IGNORESEQ to ignore changes since the list was last read.

Slot is the slot number corresponding to the job as previously returned by gspool_joblist or gspool_jobfindslot.

Jobd is a descriptor containing the job descriptor as defined in gspool.h.

The fields in struct apispq are defined in the gspool_jobread documentation (see section 6.2.2).

Note that we recommend that the whole structure be first read in with gspool_jobread or gspool_jobfind and then required fields updated; this approach will cover any future extensions with additional fields.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to change the name of job "readme.txt" to "myfile"

```
int fd, ret, nj, i;
struct apispq job;
slotno_t *slots;
```

```
fd = gspool_open("myhost", (char *) 0, 0);
if  (fd < 0) { /* error handling */
     ...
}

/* make a list of jobs */

ret = gspool_joblist(fd, 0, &nj, &slots);
if (ret < 0) { /* error handling */
     ...
}

for (i = 0; i < nj; i++) {
     ret = gspool_jobread(fd, 0, list[i], &job);
     if  (ret < 0)
         continue;
     if (strcmp(job.apispq_file, "readme.txt"))
         continue;
     strcpy(job.apispq_file, "myfile");
     ret = gspool_jobupd(fd, 0, list[i], &job);
     if (ret < 0) { /* error handling */
         ...
     }
     break;
}
gspool_close(fd);
```

6.2.12 gspool_jobmon (Unix and GNU/Linux versions)

```
int gspool_jobmon(const int fd,
            void (*fn)(const int))
```

The gspool_jobmon function is used to set the function fn to be called upon notification of any changes to the jobs list.

Fd is a file descriptor previously returned by gspool_open.

Fn is a function which must be declared as returning void and taking one const int argument. Alternatively, this may be NULL to cancel monitoring.

The function fn will be called upon each change to the job list. The argument passed will be fd. Note that any changes to the job queue are reported (including changes on other hosts whose details are passed through) as the API does not record which jobs the user is interested in.

The function gspool_jobmon returns 0 if successful otherwise the error code GSPOOL_INVALID_FD if the file descriptor is invalid. Invalid fn parameters will not be detected and the application program will probably crash.

6.2.13 gspool_jobmon (Windows version)

```
int gspool_setmon(const int fd,
            HWND hWnd,
```

```
        UINT wMsg)

  int gspool_procmon(const int fd)

  void gspool_unsetmon(const int fd)
```

The gspool_setmon routine may be used to monitor changes to the job queue or printer list. Its parameters are as follows.

fd is a file descriptor previously returned by gspool_open.

hWnd is a windows handle to which messages should be sent.

wMsg is the message id to be passed to the window (WM_USER or a constant based on this is suggested).

To decode the message, the gspool_procmon is provided. This returns XTWINAPI_JOBPROD to indicate a change or changes to the job queue and XTWINAPI_PTRPROD to indicate a change or changes to the printer list. If there are changes to both, two or more messages will be sent, each of which should be decoded via separate gspool_procmon calls.

To cancel monitoring, invoke the routine

```
  gspool_unsetmon(fd)
```

If no monitoring is in progress, or the descriptor is invalid, this call is just ignored.

6.3 Printer operations

6.3.1 gspool_ptrlist

```
  int gspool_ptrlist(const int fd,
                const unsigned flags,
                int *numptrs,
                slotno_t **slots)
```

The gspool_ptrlist function is used to obtain a list of printers.

Fd is a file descriptor previously returned by gspool_open

Flags is either zero, or GSPOOL_FLAG_LOCALONLY to request that only printers local to the server be listed.

Numptrs is a pointer to an integer value which, on successful completion, will contain the number of printer slots returned.

Slots is a pointer to to an array of slot numbers. These slot numbers can be used to access individual printers. The memory used by this vector is owned by the API, therefore no attempt should be made by the user to free it. This contrasts, for example, with X library routines.

Also note that certain other calls to the API, notably gspool_joblist, with the same fd, may reuse the space, so the contents should be copied if required before other API calls are made.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to list all printers

```
int fd, ret, np, i;
slotno_t *slots;
fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}
ret = gspool_ptrlist(fd, 0, &np, &slots);
if (ret < 0) { /* error handling */
    ...
}
for (i = 0; i < np; i++) {
    slotno_t this_slot = slots[i];
    /* process this_slot */
    ...
}
gspool_close(fd);
```

6.3.2 gspool_ptrread

```
int gspool_ptrread(const int fd,
                   const unsigned flags,
                   const slotno_t slot,
                   struct apispptr *ptrd)
```

The gspool_ptrread function is used to retrieve the details of a printer from a given slot number.

Fd is a file descriptor previously returned by gspool_open

Flags is zero, or a logical OR of one of the following values

GSPOOL_FLAG_LOCALONLY	Ignore remote printers/hosts, i.e. not local to the server, not the client.
GSPOOL_FLAG_USERONLY	Ignore other users jobs
GSPOOL_FLAG_IGNORESEQ	Ignore changes since the list was last read

Slot is the slot number corresponding to the printer as previously returned by a call to gspool_ptrlist or gspool_ptrfindslot.

Ptrd is a descriptor, which on return will contain the details of the printer in a struct apispptr as defined in gspool.h and containing the following elements:

Type	Field	Description
jobno_t	apispp_job	Job number being printed
slotno_t	apispp_jslot	Slot number of job being printed
char	apispp_state	State of printer

char	apispp_sflags	Scheduler flags
unsigned char	apispp_dflags	Despooler flags
unsigned char	apispp_netflags	Network flags
unsigned short	apispp_extrn	External printer type 0=standard
classcode_t	apispp_class	Class code bits 1=A 2=B 4=C etc
int_pid_t	apispp_pid	Process id of despooler process
netid_t	apispp_netid	Host id of printer network byte order
slotno_t	apispp_rslot	Slot number on remote machine
unsigned long	apispp_minsize	Minimum size of acceptable job
unsigned long	apispp_maxsize	Maximum size of acceptable job
char []	apispp_dev	Device name
char []	apispp_form	Form type
char []	apispp_ptr	Printer name
char []	apispp_feedback	Feedback message
char []	apispp_comment	Printer description

The following bits are set in the apispp_sflags field to indicate printer flags:

Bit (#define)	Meaning
APISPP_INTER	Had interrupt message, not yet acted on it.
APISPP_HEOJ	Had halt at end of job

The following bits are set in the apispp_dflags field to indicate printer flags:

Bit (#define)	Meaning
APISPP_HADAB	Had "Abort" message
APISPP_REQALIGN	Alignment required

The apispp_netflags field contains the following bits:

Bit (#define)	Meaning
APISPP_LOCALONLY	Printer is local only to host.
APISPP_LOCALHOST	Printer uses network filter

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to read the names of all printers

```
int fd, ret, np, i;
struct apispptr ptr;
slotno_t *slots;
fd = gspool_open("myhost", (char *)0, 0);
if (fd < 0) {  /* error handling */
    ...
}
ret = gspool_ptrlist(fd, 0, &np, &slots);
if (ret < 0) { /* error handling */
```

```
      ...
  }
  for (i = 0; i < np, i++) {
      ret = gspool_ptrread(fd, GSPOOL_FLAG_IGNORESEQ, slots[i], &ptr);
      if (ret < 0)        { /* error handling */
          ...
      }
      printf("%s\n", ptr.apispp_ptr);
  }
  gspool_close(fd);
```

6.3.3 gspool_ptrfind

```
  int gspool_ptrfind(const int fd,
                const unsigned flags,
                const char *name,
                const netid_t nid,
                slotno_t *slot,
                struct apispptr *ptrd)

  int gspool_ptrfindslot(const int fd,
                  const unsigned flags,
                  const char *name,
                  const netid_t nid,
                  slotno_t *slot)
```

The gspool_ptrfind and gspool_ptrfindslot functions may be used to find a printer from a given printer name rather than by the slot number. gspool_ptrfind retrieves the printer description, gspool_ptrfindslot just retrieves the slot number.

Fd is a file descriptor previously returned by gspool_open.

Flags is zero, or GSPOOL_FLAG_LOCALONLY to ignore remote printers/hosts, i.e. not local to the server, not the client.

Name is the printer name to be searched for.

Nid is the network-byte order IP address of the host of the machine whose printer is to be searched for. This should be correct even if GSPOOL_FLAG_LOCALONLY is specified.

Slot is a pointer to a location in which the slot number of the printer is placed if the search is successful. It may be NULL if this is not required (but this would be almost pointless for gspool_ptrfindslot).

Ptrd is a pointer to a field to contain the printer name as defined in gspool.h.

The fields in struct apispptr are defined in the gspool_ptrread documentation in section 6.3.2.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

NB If two or more printers on the same host have the same name, then it is not defined which is returned by gspool_ptrfind and gspool_ptrfindslot. In such cases, the

whole printer list should be read and the correct one selected.

6.3.4 gspool_ptradd

```
int gspool_ptradd(const int fd,
              struct apispptr *ptrd)
```

The function gspool_ptradd is used to create a printer.

Fd is a file descriptor previously returned by gspool_open.

Ptrd is a struct apispptr describing the details of the printer. It is defined in the file gspool.h and as described in the gspool_ptrread documentation in section 6.3.2.

Only values for the name, device, formtype, description, local flag, the minimum and maximum job sizes, the network filter flag and the class code are accepted. All other parameters are ignored. We suggest that you clear all fields to zero before starting. Future releases with additional fields will be guaranteed to default to the existing behaviour if the additional fields are set to zero.

gspool_ptradd returns zero if successful, otherwise an error code as listed in section 3.1.

An example to add a printer called hplj1 on device /dev/tty12 with form type a4

```
int fd, ret;
struct apispptr ptr;
fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}
memset((void *) &ptr, '\0', sizeof(ptr));
ptr.apispp_class = 0xffffffff;
ptr.apispp_minsize = ptr.apispp_maxsize = 0;
strcpy(ptr.apispp_ptr, "hplj1");
strcpy(ptr.apispp_form, "a4");
strcpy(ptr.apispp_dev, "tty12");
strcpy(ptr.apispp_comment, "My new printer");
ret = gspool_ptradd(fd, &ptr);
if (ret < 0) { /* error handling */
    ...
}
gspool_close(fd);
```

6.3.5 gspool_ptrdel

```
int gspool_ptrdel(const int fd,
              const unsigned flags,
              const slotno_t slot)
```

The function gspool_ptrdel is used to delete a printer.

Fd is a file descriptor previously returned by gspool_open.

Flags is either zero, or GSPOOL_FLAG_IGNORESEQ to ignore changes since the list was

last read.

Slot is the slot number corresponding to the printer as previously returned by gspool_ptrlist or gspool_ptrfindslot.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to delete all printers:

```
int fd, ret, np, i;
slotno_t *slots;

fd = gspool_open("myhost", (char *)0, 0);
if (fd < 0) { /* error handling */
    ...
}
ret = gspool_ptrlist(fd, GSPOOL_LOCALONLY, &np, &slots);
if (ret < 0) { /* error handling */
    ...
}

for (i = 0; i < np; i++) {
    ret = gspool_ptrdel(fd, 0, slots[i]);
    if (ret < 0) { /* error handling */
        ...
    }
}

gspool_close(fd);
```

6.3.6 gspool_ptrupd

```
int gspool_ptrupd(const int fd,
            const unsigned flags,
            const slotno_t slot,
            struct apispp *ptrd)
```

The gspool_ptrupd function is used to update the details of a printer.

Fd is a file descriptor previously returned by gspool_open.

Flags is zero, or GSPOOL_FLAG_IGNORESEQ to ignore changes since the list was last read.

Slot is the slot number corresponding to the printer as previously returned by gspool_ptrlist or gspool_ptrfindslot.

Ptrd is a descriptor containing the printer descriptor as defined in gspool.h.

The fields in struct apispptr are defined in the gspool_ptrread documentation.

Note that we recommend that the whole structure be first read in with gspool_ptrread or gspool_ptrfind and then required fields updaated; this approach will cover any future extensions with additional fields.

Only changes to the name device, description, form type, local flag, the minimun and maximum job sizes, the network filter flag and the class code are accepted, and none at all if the printer is running.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to change the form type on printer `hplj1`.

```
int fd, ret;
struct apispptr ptr;
slotno_t pslot;

fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}

/* Find printer */
ret = gspool_ptrfind(fd, 0, "hplj1", servip, &pslot, &ptr);
if (ret < 0)  { /* error handling */
    ...
}

strcpy(ptr.apispp_form, "a4.p10");
ret = gspool_ptrupd(fd, 0, pslot, &ptr);

if (ret < 0) { /* error handling */
    ...
}

gspool_close(fd);
```

6.3.7 gspool_ptrop

```
int gspool_ptrop(const int fd,
            const unsigned flags,
            const slotno_t slot,
            const unsigned op)
```

The gspool_ptrop function is used to perform an operation on a printer.

Fd is a file descriptor previously returned by gspool_open.

Flags is zero, or GSPOOL_FLAG_IGNORESEQ to ignore changes since the list was last read.

Slot is the slot number corresponding to the printer as previously returned by a call to gspool_ptrlist or gspool_ptrfindslot.

Op is one of the following values:

Operation code	Description
PRINOP_RSP	Restart printer

Operation code	Description
PRINOP_PHLT	Halt printer at the end of the current job
PRINOP_PSTP	Halt printer at once
PRINOP_PGO	Start printer
PRINOP_OYES	Approve alignment page
PRINOP_ONO	Disapprove alignment page
PRINOP_INTER	Interrupt printer
PRINOP_PJAB	Abort current job on printer

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to halt all printers:

```
int     fd, ret, np, i;
struct apispptr ptr;
slotno_t *slots;

fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}

/* make a list of all the printers */
ret = gspool_ptrlist(fd, 0, &np, &slots);
if (ret < 0) { /* error handling */
      ...
}

for (i = 0; i < np; i++) {
    ret = gspool_ptrop(fd, GSPOOL_FLAG_IGNORESEQ, slots[i],
PRINOP_PHLT);
    if (ret < 0  &&  ret != GSPOOL_PTR_NOTRUNNING) {
        /* error handling ignoring ones already stopped*/
        ...
    }
}
gspool_close(fd);
```

6.3.8 gspool_ptrmon

```
int gspool_ptrmon(const int fd,
            void (*fn)(const int))
```

NB that this routine is not available in the Windows version, please see the section on gspool_setmon in section 6.2.13 which covers both jobs and printers.

The gspool_ptrmon function is used to set the function fn to be called upon notification of any changes to the printers list.

`Fd` is a file descriptor previously returned by gspool_open

`Fn` is a function which must be declared as returning void and taking one `const int` argument. Alternatively, this may be NULL to cancel monitoring.

The function `fn` will be called with `fd` as an argument upon each change to the printer list.

Please note that any changes to the printer list is reported as the API does not record which printers the user is interested in.

The function `gspool_ptrmon` returns 0 if successful otherwise the error code `GSPOOL_INVALID_FD` if the file descriptor is invalid. Invalid `fn` parameters will not be detected and the application program will probably crash.

6.4 User permissions

The following routines access user permissions (in most cases the user will need to have write administration file privilege).

6.4.1 gspool_getspu

```
int getspu(const int fd,
           const char *user,
           struct apispdet *res)
```

The function `gspool_getspu` is used to retrieve the defaults for a particular user. Unless the calling user has Write Administration File privilege, the user name must be the calling user.

`Fd` is a file descriptor previously returned by gspool_open

`User` is a pointer to the username of the user details being retrieved.

`Res` is a descriptor, which upon return will contain the details of user. The structure `apispdet` is defined in the file `gspool.h`, and contains the following fields:

Type	Field	Description
`unsigned char`	`spu_isvalid`	Valid user ID
`char []`	`spu_resvd1`	Reserved
`int_ugid_t`	`spu_user`	User ID
`unsigned char`	`spu_minp`	Minimum priority
`unsigned char`	`spu_maxp`	Maximum priority
`unsigned char`	`spu_defp`	Default priority
`char []`	`spu_form`	Default form type
`char []`	`spu_formallow`	Allowed form type pattern
`char []`	`spu_ptr`	Default printer
`char []`	`spu_ptrallow`	Allowed printer pattern
`unsigned long`	`spu_flgs`	Privilege flag
`classcode_t`	`spu_class`	Class of printers

`unsigned char spu_cps`	Maximum copies allowed
`unsigned char spu_version`	Release of GNUspool

The `spu_flgs` field of `res` will contain a combination of the following:

`PV_ADMIN`	Administrator (edit admin file)
`PV_SSTOP`	Can run sstop (can stop scheduler)
`PV_FORMS`	Can use other forms than default
`PV_CPRIO`	Can change priority on queue
`PV_OTHERJ`	Can change other users' jobs
`PV_PRINQ`	Can move to printer queue
`PV_HALTGO`	Can halt, restart printer
`PV_ANYPRIO`	Can set any priority on queue
`PV_CDEFLT`	Can change own default priority
`PV_ADDDEL`	Can add/delete printers
`PV_COVER`	Can override class
`PV_UNQUEUE`	Can unqueue jobs
`PV_VOTHERJ`	Can view other jobs not neccesarily edit
`PV_REMOTEJ`	Can access remote jobs
`PV_REMOTEP`	Can access remote printers
`PV_FREEZEOK`	Can save default options
`PV_ACCESSOK`	Can access sub-screens
`PV_OTHERP`	Can use other printers from default
`ALLPRIVS`	A combination of all of the above

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to view the privileges of user `mark`:

```
int     fd, ret;
struct apispdet res;
fd = gspool_open("myhost", (char *)0, 0);
if (fd < 0) { /* error handling */
    ...
}
ret = gspool_getspu(fd, "mark", &res);
if (ret < 0) { /* error handling */
    ...
}
if (res.spu_flags & PV_HALTGO)
    printf("user mark cannot halt printers\n");

printf("marks maximim priority is %d\n", res.spu_maxp);
gspool_close(fd);
```

6.4.2 gspool_getspd

```
int gspool_getspd(const int fd,
                  struct apisphdr *res)
```

The `gspool_getspd` function is used to retrieve the defaults privileges, form types etc for new users on the host with which the API is communicating. No particular privilege is required to perform this operation.

`Fd` is a file descriptor previously returned by `gspool_open`.

`Res` is a descriptor which upon return will contain the the default user privileges. The structure `apisphdr` is defined in `gspool.h` and contains the following elements:

Type	Field	Description
long	sph_lastp	Time last read password file
unsigned char	sph_minp	Minimum priority
unsigned char	sph_maxp	Maximum priority
unsigned char	sph_defp	Default priority
char []	sph_form	Default form type
char []	sph_formallow	Allowed form type pattern
char []	sph_ptr	Default printer
char []	sph_ptrallow	Allowed printer pattern
unsigned long	sph_flgs	Privilege flag
classcode_t	sph_class	Class of printers
unsigned char	sph_cps	Maximum copies allowed
unsigned char	sph_version	Release of GNUspool

The `spu_flgs` field will contain a combination of the following:

PV_ADMIN	Administrator (edit admin file)
PV_SSTOP	Can run sstop (can stop scheduler)
PV_FORMS	Can use other forms than default
PV_CPRIO	Can change priority on queue
PV_OTHERJ	Can change other users' jobs
PV_PRINQ	Can move to printer queue
PV_HALTGO	Can halt, restart printer
PV_ANYPRIO	Can set any priority on queue
PV_CDEFLT	Can change own default priority
PV_ADDDEL	Can add/delete printers
PV_COVER	Can override class
PV_UNQUEUE	Can unqueue jobs
PV_VOTHERJ	Can view other jobs not necessarily edit
PV_REMOTEJ	Can access remote jobs
PV_REMOTEP	Can access remote printers
PV_FREEZEOK	Can save default options
PV_ACCESSOK	Can access sub-screens

PV_OTHERP   Can use other printers from default

ALLPRIVS   A combination of all of the above

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to view the default privileges on the host machine:

```
int fd, ret;
struct apisphdr res;

fd = gspool_open("myhost", (char *) 0, 0);
if (fd < 0) { /* error handling */
    ...
}

ret = gspool_getspd(fd, &res);
if (ret < 0)  { /* error handling */
    ...
}

if (res.sph_flgs & PV_HALTGO)
    printf("users cannot stop and start printers\n");
printf("the default maximum priority is %s\n", res);
gspool_close(fd);
```

6.4.3 gspool_putspu

```
int gspool_putspu(const int fd,
              const char *user,
              struct apispdet *newp)
```

The gspool_putspu function is used to set privileges for a user. The calling user must have write administration file privilege, or must be the same as the specified user and be only trying to change the default form type or priorities (with the appropriate privilege for that).

Fd is a file descriptor previously returned by gspool_open.

User is a pointer to the user name, for which the details are being updated.

Newp is a pointer to a structure containing the new user privileges.

The struct apispdet is defined int the file gspool.h. The fields of the structure are as defined for gspool_getspu in section 6.4.1.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to give a user permission to add and delete printers

```
int fd, ret;
struct apispdet new_privs;

fd = gspool_open("myhost", (char *)0, 0);
```

```
if (fd < 0) { /* error handling */
    ...
}

ret = gspool_getspu(fd, "helen", &new_privs);
if (ret < 0)   { /* error handling */
    ...
}

if (!(new_privs.spu_flgs & PV_ADDDEL))
    new_privs.spu_flgs |= PV_ADDDEL;

gspool_close(fd);
```

6.4.4 gspool_putspd

```
int gspool_putspd(const int fd,
              struct apisphdr *ret)
```

The gspool_putspd function is used to set the default user privileges on the local host.

Its parameters are as follows:

Fd is a file descriptor previously returned by gspool_open.

Res points to a structure which contains the privileges. The struct apisphdr is defined in the file gspool.h, as described for gspool_getspd in section 6.4.2.

The function returns 0 if successful otherwise one of the error codes as listed in section 3.1.

An example to give all new users the permission to add and delete printers:

```
int fd, ret;
struct apisphdr new_privs;
fd = gspool_open("myhost", (char *)0, 0);
if (fd < 0) {  /* error handling */
    ...
}
/* get the current permissions */
ret = gspool_getspd(fd, &new_privs);
if (ret < 0) { /* error handling */
    ...
}
if (!(new_privs.sph_flgs & PV_ADDDEL))
    new_privs.sph_flgs |= PV_ADDDEL;
ret = gspool_putspd(fd, &new_privs);
if (ret < 0) { /* error handling */
    ...
}
gspool_close(fd);
```

7 Example API program

The following program is an example program to provide for an "alternative printer" to be activated when a machine running the main printer is or goes offline. The program runs on the "secondary" machine:

```
/*
 * altprin.c: created by John Collins.
 */

#include <stdio.h>
#include <sys/types.h>
#include "gspool.h"
#include <unistd.h>
#include <netdb.h>
#include <string.h>

int  had_prod;
char *primary,            /* Primary host name */
     *secondary,          /* Secondary host name */
     *primary_prin,       /* Primary printer name */
     *secondary_prin;     /* Secondary printer name */

netid_t  prim_hostid, sec_hostid;

int  xtfd;

/*
 *    Routine to call when printer event occurs.
 *    Just set flag and let the main loop look at it
 *    when it is ready.
 */

void    prodder(const int fd)
{
    had_prod++;
}

void  process(void)
{
    /*
     *  Say we want to know about events affecting printers.
     */

    gspool_ptrmon(xtfd, prodder);

 gotpri:
    for (;;)  {
        int     nump, cnt, ret;
        slotno_t *slp;
        struct  apispptr res;

        /*
         *  Wait until something interesting happens to a printer.
         */
```

```
        pause();
        if (!had_prod)  /* Huh??? */
            continue;
        had_prod = 0;

        /*
         *  Get list of printers "slot numbers" into "slp", number
         *  into "nump".
         *  We don't really need to do this on each loop if printer
         *  slot numbers don't change too much, which they don't
         */

        if (gspool_ptrlist(xtfd, 0, &nump, &slp) < 0)
            exit(255);

        /*
         *  Search list for primary printer.
         *  If found, all is ok, and we go back to sleep.
         */

        for (cnt = 0;  cnt < nump;  cnt++)  {
            if (gspool_ptrread(xtfd,
                               GSPOOL_FLAG_IGNORESEQ,
                               slp[cnt],
                               &res) < 0)
                exit(254);
            if  (res.apispp_netid != prim_hostid)
                continue;
            if  (strcmp(res.apispp_ptr, primary_prin) == 0)
                goto  gotpri;
        }

        /*
         *  We didn't find primary printer, so we start up the
         *  secondary printer. First find the thing.
         */

        for (cnt = 0;  cnt < nump;  cnt++)  {
            if (gspool_ptrread(xtfd,
                               GSPOOL_FLAG_IGNORESEQ,
                               slp[cnt],
                               &res) < 0)
                exit(254);
            if (res.apispp_netid != sec_hostid)
                continue;
            if (strcmp(res.apispp_ptr, secondary_prin) == 0)
                goto  gotsec;
        }
        fprintf(stderr, "Cannot find secondary printer, %s\n",
secondary_prin);
        exit(200);

        /*
         *  Found secondary printer, print a warning message
         *  if already running.
         */
```

```
gotsec:
        if (res.apispp_state >= API_PRPROC) {
            fprintf(stderr,
          "I think that the secondary printer is already running\n");
            exit(0);
        }

        /*
         *  Tell the world, start it up, and exit
         */

        fprintf(stderr, "Activating secondary printer
                    %s:%s\n", secondary, secondary_prin);

        if ((ret = gspool_ptrop(xtfd, GSPOOL_FLAG_IGNORESEQ, slp[cnt],
PRINOP_PGO)) < 0) {
            printf("Error starting printer - %d\n", ret);
            exit(0);
        }
}

int main(int argc, char **argv)
{
    extern  intoptind;
    extern  char    *optarg;
    int     ch;
    struct  hostent*hp;
    char        *cp;
    static  char    myname[256];

    /*
     *  Get "my" host name.
     */

    myname[sizeof(myname) - 1] = '\0';
    gethostname(myname, sizeof(myname) - 1);
    if (!(hp = gethostbyname(myname))) {
        fprintf(stderr, "Who am I???\n");
        return 10;
    }

    /*
     *  Get arguments giving primary and secondary printers.
     */

    while ((ch = getopt(argc, argv, "p:s:")) != EOF)  {
        switch  (ch)  {
        default:
            fprintf(stderr,
                "Usage: altprin -p primary -s secondary\n");
            return  1;
        case 'p':
            primary = optarg;
            break;
        case 's':
            secondary = optarg;
            break;
```

```
    }
}

if (!primary)  {
    fprintf(stderr, "No primary host:printer name given\n");
    return 2;
}
if (!secondary)  {
    fprintf(stderr, "No secondary host:printer name given\n");
    return 3;
}

/*
 *  Split host:printer names into separate strings.
 *  If not host name, tack on "my" name.
 */

if (cp = strchr(primary, ':'))  {
    *cp = '\0';
    primary_prin = cp+1;
}
else {
    primary_prin = primary;
    primary = myname;
    fprintf(stderr, "Primary printer on local host?\n");
}
if (cp = strchr(secondary, ':'))  {
    *cp = '\0';
    secondary_prin = cp+1;
}
else {
    secondary_prin = secondary;
    secondary = myname;
}

if (strcmp(primary, secondary) == 0) {
    fprintf(stderr, "Sorry both printers on the same host\n");
    return  4;
}

/*
 *  Get host ids, used in scanning printer list.
 */

if (!(hp = gethostbyname(primary))) {
    fprintf(stderr, "Sorry, unknown primary host name %s\n",
                    primary);
    return  5;
}
else
    prim_hostid = *(netid_t *) hp->h_addr;

if (!(hp = gethostbyname(secondary))) {
    fprintf(stderr, "Sorry, unknown secondary host name %s\n",
            secondary);
    return  6;
}
```

```
    else
        sec_hostid = *(netid_t *) hp->h_addr;

    /*
     *  Open API link.
     */

    if ((xtfd = gspool_open(secondary, (char *) 0, 0)) < 0)  {
        fprintf(stderr,
            "Sorry, cannot open connection to secondary host\n");
        return 7;
    }

    /*
     *  Fork off to leave a daemon process.
     *  (You might want to set process group, ignore
     *  signals and/or reconnect
     *  stdout/stderr).
     */

    if (fork() != 0)
        return 0;

    /*
     *  Do the business (no return).
     */

    process();
}
```