Open Standard Print API (PAPI)

Version 0.2 (DRAFT)
April 17, 2002

Alan Hlava, IBM Printing Systems Division
Michael R. Sweet, Easy Software Products

1. *Introduction*

This document describes the Open Standard Print Application Programming Interface (API), also known as "PAPI" (Print API). This is a set of open standard C functions that can be called by application programs to use the print spooling facilities available in Linux (NOTE: this interface is being proposed as a print standard for Linux, but there is really nothing Linux-specific about it and it could be adopted on other platforms). Typically, the "application" is a GUI program attempting to perform a request by the user to print something.

This version of the document describes stage 1 and stage 2 of the Open Standard Print API:

Stage 1: Simple interfaces for job submission and querying printer capabilities (attributes)
Stage 2: Addition of interfaces to use Job Tickets, addition of operator interfaces
Stage 3: Addition of administrative interfaces (create/delete objects, enable/disable objects, etc.)

Subsequent versions of this document will incorporate the additional functions described in the later stages.

2. *Print System Model*

2.1. Introduction

Any printing system API must be based on some "model". A printing system model defines the objects on which the API functions operate (e.g. a "printer"), and how those objects are interrelated (e.g. submitting a file to a "printer" results in a "job" being created).

The print system model must answer the following questions in order to be used to define a set of print system APIs:

- Object Definition: What objects are part of the model?
- Object Naming: How is each object identified/named?
- Object Relationships: What are the associations and relationships between the objects?

Some examples of possible objects a printing system model might include are:

Printer	Document	Medium/Form
Server	Queue	Filter/Transform
Job	Class/Pool	Print Resource (font, etc.)
Job Ticket	Auxiliary Sheet	

2.2. Model

The model on which the Open Standard Print API will be based is the one defined by the Internet Print Protocol (IPP) standard. This is a fairly simple model in terms of the number of object types. It is defined very clearly and in detail in the IPP RFC 2911, Chapter 2 (http://ietf.org/rfc/rfc2911.txt?number=2911).

Consult the above document for a thorough understanding of the IPP print model. A quick summary of the model is provided here.

2.2.1. Printer

Printer objects are the target of print job requests. A printer object may represent an actual printer (if the printer itself supports IPP), an object in a server representing an actual printer, or an abstract object in a server (perhaps representing a pool or class of printers). Printer objects are identified via one or more URIs, such as "http://printserv.mycompany.com:631/printers/prtr1". A printer object may also be identified via a local (to the device/server) "printer-name".

2.2.2. Job

Job objects are created after a successful print submission. They contain a set of attributes describing the job and specifying how it will be printed, and they contain (logically) the print data itself in the form of one or more "documents". Job objects are identified via a URI, such as "http://printserv.mycompany.com:631/jobs/178". A job object may also be identified via a local (to the device/server) "job-id".

The local printer-name and local job-id cannot be used by themselves to identify these objects on the API since they are not globally unique. However, a directory service may be layered on top of the API which could map local names to global URIs. As far as the API is concerned, the local names are for informational and display purposes only.

2.3.Security

The security model of this API is based on the IPP security model, which uses HTTP security mechanisms.

2.3.1. Authentication

Either HTTP Basic authentication or HTTP Digest authentication may be used, depending on the capabilities and configuration of the server/printer being used. In either case, a user name and password should be provided on the request. If HTTP Basic authentication is used then the user name and password are passed with the request Base64-encoded, which if HTTP Digest authentication is used then an MD5 checksum of the user name and password are passed instead of the strings.

If the user name and password are not passed on the API call, the call may fail with an error code indicating a security problem (e.g. PAPI_NOT_AUTHENTICATED).

See RFC 2616 and RFC 2617 for further details about HTTP security.

2.3.2. Authorization

Authorization is the security checking that follows authentication. It verifies that the identified user is authorized to perform the requested operation on the specified object.

Since authorization is an entirely server-side (or printer-side) function, how it works is not specified by this API. In other words, the server (or printer) may or may not do authorization checking according to its capability and current configuration. If authorization checking is performed, any call may fail with an error code indicating the failure (PAPI_NOT_AUTHORIZED).

2.3.3. Encryption

Encrypting certain data sent to and from the printer/server may be desirable in some environments. See field "encryption" in section "3.1. Object Identification (papi_object_id_t)" for how to request encryption on a print operation. Note that some printers/servers may not support encryption. To comply with this standard, only the HTTP_ENCRYPT_NEVER value must be supported.

3. Common Structures

The following C structures are used by the APIs

3.1. Object Identification (papi_object_id_t)

This structure is used to identify which object a request will be performed on.

```
typedef struct
{
   char  hostname[HTTP_MAX_HOST]; /* Hostname/IP Addr thru which
                                     object is to be accessed */
   int         port;              /* Port on hostname to use   */
   http_encryption_t encryption; /* Encryption requirements */
   char        uri[HTTP_MAX_URI]; /* URI of object */
   char        short_name[PAPI_MAX_NAME]; /* Short name of object */
   http_t*     http;              /* Open connection (optional) */
} papi_object_id_t;
```

Some implementations may support identifying the object via a short name instead of a URI (although this is not required). If so, the object may be identified via the "short_name" field instead of the "uri" field. If the "uri" field is specified, then "short_name" is ignored.

The "http" field is only used if papiOpenConnection/papiCloseConnection are called to keep a connection open across multiple API calls. Otherwise, it must be set to NULL.
The "encryption" field may be set to one of:

```
typedef enum
{
  HTTP_ENCRYPT_IF_REQUESTED,/* Encrypt if requested (TLS upgrade) */
  HTTP_ENCRYPT_NEVER,       /* Never encrypt */
  HTTP_ENCRYPT_REQUIRED,    /* Encryption is required (TLS upgrade) */
  HTTP_ENCRYPT_ALWAYS       /* Always encrypt (SSL) */
} http_encryption_t;
```

Note that to comply with this standard, only the HTTP_ENCRYPT_NEVER value must be supported.

3.2. Common Arguments (papi_common_args_t)

This structure is used to identify common arguments which are used on all API calls.

```
typedef struct
{
   char  language[PAPI_MAX_NAME];
                            /* attributes-natural-language
                              (see IPP RFC 2911 for details)
                               If empty, defaults based on
                               The current locale */
   char* char_set[PAPI_MAX_NAME];
                            /* attributes-character-set
                               (see IPP RFC 2911 for details)
                               If empty, defaults to "utf-8" */
   char* user_name[PAPI_MAX_NAME];
                            /* requesting-user-name
                               (see IPP RFC 2911 for details)
                               If empty, tries to determine
                               from operating system */
   char* password[PAPI_MAX_NAME];
                            /* password for user authentication */
} papi_common_args_t;

#define PAPI_INIT_COMMON_ARGS(x) \
{ \
   x.language[0]  = '\0'; \
   x.char_set[0]  = '\0'; \
   x.user_name[0] = '\0'; \
   x.password[0]  = '\0'; \
}
```

3.3. Result (papi_result_t)

This structure is used to return the results of an operation to the caller. This is an opaque data structure that the API calls allocate on the heap and return to the caller. The caller will then pass this structure to other APIs to check for errors and retrieve data from the results. When done with it, it is the caller's responsibility to delete the results structure by calling the papiDelete API.

3.4.Status (papi_status_t)

```
typedef enum
{
  PAPI_OK = 0x0000,
  PAPI_OK_SUBST,
  PAPI_OK_CONFLICT,
  PAPI_OK_IGNORED_SUBSCRIPTIONS,
  PAPI_OK_IGNORED_NOTIFICATIONS,
  PAPI_OK_TOO_MANY_EVENTS,
  PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
  PAPI_REDIRECTION_OTHER_SITE = 0x300,
  PAPI_BAD_REQUEST = 0x0400,
  PAPI_FORBIDDEN,
  PAPI_NOT_AUTHENTICATED,
  PAPI_NOT_AUTHORIZED,
  PAPI_NOT_POSSIBLE,
  PAPI_TIMEOUT,
  PAPI_NOT_FOUND,
  PAPI_GONE,
  PAPI_REQUEST_ENTITY,
  PAPI_REQUEST_VALUE,
  PAPI_DOCUMENT_FORMAT,
  PAPI_ATTRIBUTES,
  PAPI_URI_SCHEME,
  PAPI_CHARSET,
  PAPI_CONFLICT,
  PAPI_COMPRESSION_NOT_SUPPORTED,
  PAPI_COMPRESSION_ERROR,
  PAPI_DOCUMENT_FORMAT_ERROR,
  PAPI_DOCUMENT_ACCESS_ERROR,
  PAPI_ATTRIBUTES_NOT_SETTABLE,
  PAPI_IGNORED_ALL_SUBSCRIPTIONS,
  PAPI_TOO_MANY_SUBSCRIPTIONS,
  PAPI_IGNORED_ALL_NOTIFICATIONS,
  PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,

  PAPI_INTERNAL_ERROR = 0x0500,
  PAPI_OPERATION_NOT_SUPPORTED,
  PAPI_SERVICE_UNAVAILABLE,
  PAPI_VERSION_NOT_SUPPORTED,
  PAPI_DEVICE_ERROR,
  PAPI_TEMPORARY_ERROR,
  PAPI_NOT_ACCEPTING,
  PAPI_PRINTER_BUSY,
  PAPI_ERROR_JOB_CANCELLED,
  PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
  PAPI_PRINTER_IS_DEACTIVATED
} papi_status_t;
```

3.5.Tag (papi_tag_t)

```
typedef enum
```

```
{
  PAPI_TAG_ZERO = 0x00,
  PAPI_TAG_OPERATION,
  PAPI_TAG_JOB,
  PAPI_TAG_END,
  PAPI_TAG_PRINTER,
  PAPI_TAG_UNSUPPORTED_GROUP,
  PAPI_TAG_SUBSCRIPTION,
  PAPI_TAG_EVENT_NOTIFICATION,
  PAPI_TAG_UNSUPPORTED_VALUE = 0x10,
  PAPI_TAG_DEFAULT,
  PAPI_TAG_UNKNOWN,
  PAPI_TAG_NOVALUE,
  PAPI_TAG_NOTSETTABLE = 0x15,
  PAPI_TAG_DELETEATTR,
  PAPI_TAG_ADMINDEFINE,
  PAPI_TAG_INTEGER = 0x21,
  PAPI_TAG_BOOLEAN,
  PAPI_TAG_ENUM,
  PAPI_TAG_STRING = 0x30,
  PAPI_TAG_DATE,
  PAPI_TAG_RESOLUTION,
  PAPI_TAG_RANGE,
  PAPI_TAG_BEGIN_COLLECTION,
  PAPI_TAG_TEXTLANG,
  PAPI_TAG_NAMELANG,
  PAPI_TAG_END_COLLECTION,
  PAPI_TAG_TEXT = 0x41,
  PAPI_TAG_NAME,
  PAPI_TAG_KEYWORD = 0x44,
  PAPI_TAG_URI,
  PAPI_TAG_URISCHEME,
  PAPI_TAG_CHARSET,
  PAPI_TAG_LANGUAGE,
  PAPI_TAG_MIMETYPE,
  PAPI_TAG_MEMBERNAME,
  PAPI_TAG_MASK = 0x7fffffff, /* Mask for copied attr values */
  PAPI_TAG_COPY = -0x7fffffff-1 /* Bitflag for copied attr values */
} papi_tag_t;
```

3.6. Resolution Units (papi_res_t)

```
typedef enum
{
  PAPI_RES_PER_INCH = 3,
  PAPI_RES_PER_CM
} papi_res_t;
```

3.7. Attribute (papi_attribute_t)

This is the structure containing an attribute and its associated value(s). This structure closely mirrors the IPP model of an attribute.

```
typedef struct papi_attribute_s
{
  struct papi_attribute_s *next; /* Next attribute in list */
  papi_tag_t   group_tag,        /* Job/Printer/Oper group tag */
         value_tag;              /* What type of value is it? */
  char         *name;            /* Name of attribute */
  int          num_values;       /* Number of values */
```

```
  papi_value_t values[1];        /* Values */
} papi_attribute_t;
```

The values are defined by the papi_value_t structure.

```
typedef union
{
  int          integer;    /* Integer/enumerated value */

  char         boolean;    /* Boolean value */

  unsigned char date[11];  /* Date/time value */

  struct
  {
    int        xres,       /* Horizontal resolution */
         yres;             /* Vertical resolution */
    papi_res_t units;      /* Resolution units */
  } resolution;            /* Resolution value */

  struct
  {
    int        lower,      /* Lower value */
         upper;            /* Upper value */
  } range;                 /* Range of integers value */

  struct
  {
    char *charset;         /* Character set */
    char *text;            /* String */
  } string;                /* String with language value */

  struct
  {
    int        length;     /* Length of attribute */
    void *data;            /* Data in attribute */
  }            unknown;    /* Unknown attribute type */
} papi_value_t;
```

The value_tag field in the attribute determines which of the values/structures in a value union that is used by that value. The value_tag may contain:

```
  PAPI_TAG_INTEGER
  PAPI_TAG_BOOLEAN
  PAPI_TAG_ENUM
  PAPI_TAG_STRING
  PAPI_TAG_DATE
  PAPI_TAG_RESOLUTION
  PAPI_TAG_RANGE
  PAPI_TAG_BEGIN_COLLECTION
  PAPI_TAG_TEXTLANG
  PAPI_TAG_NAMELANG
  PAPI_TAG_END_COLLECTION
  PAPI_TAG_TEXT
  PAPI_TAG_NAME
  PAPI_TAG_KEYWORD
  PAPI_TAG_URI
  PAPI_TAG_URISCHEME
  PAPI_TAG_CHARSET
  PAPI_TAG_LANGUAGE
  PAPI_TAG_MIMETYPE
  PAPI_TAG_MEMBERNAME
```

For the valid attribute names which may be supported, see IPP RFC 2911.

You may iterate through returned attributes using papiGetNextAttribute and an iterator:

```
typedef void* papi_attr_iterator_t;
```

3.8.Job Ticket (papi_job_ticket_t)

This is the structure used to pass a job ticket when submitting a print job. Currently, Job Definition Format (JDF) is the only supported job ticket format. JDF is an XML-based job ticket syntax. The JDF specification can be found at www.cip4.org.

```
typedef enum
{
  PAPI_JT_FORMAT_JDF = 0,        /* Job Definition Format */
} papi_jt_format_t;

typedef struct papi_job_ticket_s
{
   papi_jt_format_t  format,     /* Format of job ticket */
   char*             ticket_data,/* Buffer containing the job
                                    ticket data.  If NULL,
                                    uri must be specified */
   char*             uri,        /* URI of the file containing
                                    the job ticket data. If
                                    ticket_data is specified, then
                                    uri is ignored. */
} papi_job_ticket_t;
```

4. Printer API

4.1. Usage

The papiQueryPrinter function queries all/some of the attributes of a printer object. It returns a results structure containing the retrieved data. A successful call to papiQueryPrinter is typically followed by calls to papiFindAttribute (to find specific attribute(s)) or to papiGetNextAttribute (to access all returned attributes). The using program would then call papiDelete to free the returned results.

Printers can be found via calls to papiListPrinters. A successful call to papiListPrinters is typically followed by calls to papiFindAttribute and papiFindNextAttribute to iterate through the list of returned printers, possibly querying each (papiQueryPrinter) for further information (e.g. to restrict what printers get displayed for a particular user/request). The using program would then call papiDelete to free the returned results.

4.2. papiListPrinters

Description: List all printers known by the specified print server.

Depending on the functionality of the target server's "printer directory", the returned list may be limited to only printers managed by that particular server or it may include printers managed by other servers.

Syntax:

```
papi_result_t* papiListPrinters(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const char* requested_attrs[],
            const int num_requested_attrs
                              );
```

Arguments:
object: Identifies the server whose printers will be listed
common_args: Common arguments
requested_attrs: Array of attributes to be queried. If NULL is passed then only the printer-uri-supported attribute is queried. (NOTE: The printer may return more attributes than you requested. This is merely an advisory request that may reduce the amount of data returned if the printer/server supports it.)
num_requested_attrs: Number of requested attributes in the requested_attrs array. If zero is passed then only the printer-uri-supported attribute is queried.

Returns: A pointer to a results structure containing returned attributes and error information (if any). If a NULL value is returned, a connection could not be made to the print server (e.g. incorrect URI, server off or offline, etc.). If the query is successful the results will contain the returned attributes from all the printers which the target server knows about. To process these, the caller would typically then use papiFindAttribute/papiFindNextAttribute to iterate through the list of returned printers.

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
papi_attribute_t* attr;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631");
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);
```

```
result = papiListPrinters(&object, &common_args, NULL, 0);
/* check for success */
…
attr = papiFindAttribute(result, “printer-uri-supported”);
while(attr != NULL)
{
      /* process this printer */
      …
      attr = papiFindNextAttribute(result,
                  “printer-uri-supported”);
}
```

See Also:
papiDelete, papiFindAttribute, papiFindNextAttribute, papiGetStatus, papiGetErrorString

4.3. papiQueryPrinter

Description: Queries some or all the attributes of the specified printer object. This includes attributes representing the capabilities of the printer, which the caller may use to determine which print options to present to the user. *How* the attributes are obtained (e.g. from a static database, from a dialog with the hardware, from a dialog with a driver, etc.) is up to the implementer of the API and is beyond the scope of this standard.

Syntax:

```
papi_result_t* papiQueryPrinter(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const char* requested_attrs[],
            const int num_requested_attrs
                              );
```

Arguments:
object: Identifies the printer to be queried
common_args: Common arguments
requested_attrs: Array of attributes to be queried. If NULL is passed then all attributes are queried. (NOTE: The printer may return more attributes than you requested. This is merely an advisory request that may reduce the amount of data returned if the printer/server supports it.)
num_requested_attrs: Number of requested attributes in the requested_attrs array. If zero is passed then all attributes are queried.

Returns: A pointer to a results structure containing error information (if any) and the printer attributes returned from the printer object. If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include “papi.h”

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/printers/prtr1”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiQueryPrinter(&object, &common_args, NULL, 0);
```

See Also:
papiDelete, papiGetStatus, papiFindAttribute, papiGetNextAttribute

4.4. papiPausePrinter

Description: Stops the printer object from scheduling jobs to be printed. Depending on the implementation, this operation may also stop the printer from processing the current job(s). This operation is optional and may not be supported by all printers/servers. Use papiResumePrinter to undo the effects of this operation.

See IPP's RFC 2911, section 3.2.7 "Pause-Printer Operation" for further details on the semantics of this operation.

Syntax:

```
papi_result_t* papiPausePrinter(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args
                                );
```

Arguments:
object: Identifies the printer to be paused
common_args: Common arguments

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631/printers/prtr1");
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiPausePrinter(&object, &common_args);
```

See Also:
papiDelete, papiGetStatus, papiResumePrinter

4.5. papiResumePrinter

Description: Requests the printer object to resume scheduling jobs to be printed (i.e. it undoes the effects of papiPausePrinter). This operation is optional and may not be supported by all printers/servers, but it must be supported if papiPausePrinter is supported.

See IPP's RFC 2911, section 3.2.8 "Resume-Printer Operation" for further details on the semantics of this operation.

Syntax:

```
papi_result_t* papiResumePrinter(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args
                              );
```

Arguments:
object: Identifies the printer to be resumed
common_args: Common arguments

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631/printers/prtr1");
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiResumePrinter(&object, &common_args);
```

See Also:
papiDelete, papiGetStatus, papiPausePrinter

4.6. papiPurgeJobs

Description: Remove all jobs from the specified printer object regardless of their states. This includes removing jobs that have completed and are being kept for history (if any). This operation is optional and may not be supported by all printers/servers.

Syntax:

```
papi_result_t* papiPurgeJobs(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args
                              );
```

Arguments:

object: Identifies the printer to purge jobs from

common_args: Common arguments

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include “papi.h”

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/printers/prtr1”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiPurgeJobs(&object, &common_args);
```

See Also:

papiDelete, papiGetStatus, papiCancelJob

5. *Attributes API*

5.1. papiFindAttribute

Description: Gets a pointer to the specified attribute in the results returned from an operation.

Syntax:

```
papi_attribute_t* papiFindAttribute(
            const papi_result_t* result,
            const char* name
                              );
```

Arguments:

result:	Pointer to result structure from which to retrieve attributes
name:	Pointer to a string containing the name of the attribute to find

Returns: A pointer to the attribute in the specified results. If a NULL value is returned, the attribute was not found in the results.

Example:

```
#include “papi.h”

papi_attribute_t* attr;
papi_result_t* result;
…
attr = papiFindAttribute(result, “printer-name”);
```

See Also:
papiFindNextAttribute, papiGetNextAttribute

5.2. papiFindNextAttribute

Description: Used after an initial call to papiFindAttribute, this function gets a pointer to the next attribute with the specified name in the results returned from an operation. This function is useful following operations that can return multiple attributes with the same name, such as papiListJobs which returns multiple job-id and job-uri attributes.

Syntax:

```
papi_attribute_t* papiFindNextAttribute(
            const papi_result_t* result,
            const char* name
                              );
```

Arguments:

result: Pointer to result structure from which to retrieve attributes
name: Pointer to a string containing the name of the attribute to find

Returns: A pointer to the next attribute in the specified results. If a NULL value is returned, no more attributes with that name were found in the results.

Example:

```
#include “papi.h”

papi_attribute_t* attr;
papi_result_t* result;
…
attr = papiFindAttribute(result, “job-uri”);
while(attr != NULL)
{
      /* process the returned attribute */
      …
      attr = papiFindNextAttribute(result, “job-uri”);
}
```

See Also:
papiFindAttribute

5.3. papiGetNextAttribute

Description: Gets a pointer to the next attribute in the results returned from an operation.

Syntax:

```
papi_attribute_t* papiGetNextAttribute(
            const papi_result_t* result,
                  papi_attr_iterator_t* iterator
                              );
```

Arguments:

result: Pointer to result structure from which to retrieve attributes

iterator: Pointer to an iterator. Set iterator to NULL to get the first attribute

Returns: A pointer to the next attribute in the specified results. If a NULL value is returned, the previous attribute was the last one in the results.

Example:

```
#include “papi.h”

papi_attribute_t* attr;
papi_attr_iterator_t iterator = NULL;
papi_result_t* result;
…
for(attr = papiGetNextAttribute(result, iterator);
    attr != NULL;
    attr = papiGetNextAttribute(result, iterator)
{
      /* process the attribute */
      …
}
```

See Also:

papiFindAttribute

6. *Job API*

6.1. papiSubmitJob

Description: Submits a print job having the specified attributes to the specified printer.

Syntax:

```
papi_result_t* papiSubmitJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const char* job_attributes,
            const papi_job_ticket_t* job_ticket,
            const char* file_names
                        );
```

Arguments:

object: Identifies the printer to which the job will be submitted

common_args: Common arguments

job_attributes:

(optional) The set of attributes describing the job and how it is to be printed. This is a string containing a sequence of attribute names and values having syntax: "name1 = value1 [value2 …[valueN]] ; name2 = …". The individual values must be enclosed in single-quotes if they contain any whitespace characters. If options are specified here and also in the job ticket data, the value specified here takes precedence. If this is NULL then only default attributes and (optionally) a job ticket is passed submitted the job.

job_ticket:

(optional) Pointer to structure specifying the JDF job ticket. If this argument is NULL, then no job ticket is used with the job.

file_names:

Names of one or more files to print (separated by semi-colons).

If the name(s) begin with one of the following strings, then they are assumed to be URIs and will be passed to the server by name: "http:", https:", "ftp:", "file:". The printer/server is expected to "pull" the actual print data across from the specified location when it is needed. This feature is optional and may not be supported by all printers/servers.

Returns: A pointer to a results structure containing error information (if any) and all the attributes returned from the job submission. If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.). If the submission is successful the results will contain, at a minimum, the job-uri, job-id, and job-state attributes.

See the IPP standard for details of what attributes must/may be returned on successful and unsuccessful job submissions (RFC 2911, 3.2.1.2 “Print-Job Response”).

Example:

```
#include “papi.h”

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
const char* job_attrs =
      “job-name = ‘My Job’; \
      copies = 3; \
      sides = two-sided-long-edge;”
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/printers/prtr1”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiSubmitJob(&object,
            &common_args,
            job_attrs,
            NULL,
            “/home/smith/myjob.ps”);
```

See Also:
papiDelete, papiGetStatus, papiGetErrorString

6.2. papiValidateJob

Description: Validates the specified job attributes against the specified printer. This function can be used to validate the capability of a print object to accept a specific combination of attributes.

Syntax:

```
papi_result_t* papiValidateJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const char* job_attributes
                        );
```

Arguments:

object: Identifies the printer against which the job will be validated

common_args: Common arguments

job_attributes:

The set of attributes describing the job and how it is to be printed. This is a string containing a sequence of attribute names and values having syntax: "name1 = value1 [value2 …[valueN]] ; name2 = …". The individual values must be enclosed in single-quotes if they contain any whitespace characters.

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.). If the submission is successful the results will contain a status code indicating success.

The status codes and attributes returned are the same as those returned from papiSubmitJob, except that no job object attributes (job-id, job-uri) are returned because no job is actually created.

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
const char* job_attrs =
      "job-name = 'My Job'; \
      copies = 3; \
      sides = 2;"
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631/printers/prtr1");
object.http = NULL;
```

```
PAPI_INIT_COMMON_ARGS(common_args);

result = papiValidateJob(&object,
            &common_args,
            job_attrs);
```

See Also:
papiSubmitJob, papiDelete, papiGetStatus, papiGetErrorString

6.3. papiListJobs

Description: List print job(s) associated with the specified printer.

Syntax:

```
papi_result_t* papiListJobs(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const char* requested_attrs[],
            const int num_requested_attrs,
            const int my_jobs,
            const int return_completed,
            const int max_num_jobs
                        );
```

Arguments:
object: Identifies the printer whose jobs will be listed
common_args: Common arguments
requested_attrs: Array of attributes to be queried. If NULL is passed then only job-id and job-uri attributes are queried. (NOTE: The printer may return more attributes than you requested. This is merely an advisory request that may reduce the amount of data returned if the printer/server supports it.)
num_requested_attrs: Number of requested attributes in the requested_attrs array. If zero is passed then only job-id and job-uri attributes are queried.
my_jobs: Indicates whether to return anyone's jobs or just the user_name's (in common_args) jobs. A 0 indicates to return anyone's jobs and a 1 indicates to return just the user_name's jobs.
return_completed: Indicates whether to return completed or non-completed (waiting and in process) jobs. A 0 indicates to return non-completed jobs and a 1 indicates to return completed jobs.
max_num_jobs: Limit to the number of jobs returned. If 0 is passed, then there is no limit on the number of jobs which may be returned.

Returns: A pointer to a results structure containing returned attributes and error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.). If the query is successful the results will contain the returned attributes from the jobs. To process these, the caller would typically then use papiFindAttribute/papiFindNextAttribute to iterate through the list of returned jobs.

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
papi_attribute_t* attr;
```

```
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/printers/prtr1”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiListJobs(&object, &common_args, NULL, 0, 0, 0, 0);
/* check for success */
…
attr = papiFindAttribute(result, “job-uri”);
while(attr != NULL)
{
      /* process this job */
      …
      attr = papiFindNextAttribute(result, “job-uri”);
}
```

See Also:
papiDelete, papiFindAttribute, papiFindNextAttribute, papiGetStatus, papiGetErrorString

6.4. papiQueryJob

Description: Queries some or all the attributes of the specified job object.

Syntax:

```
papi_result_t* papiQueryJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const int job_id,
            const char* requested_attrs[],
            const int num_requested_attrs
                        );
```

Arguments:
object: Identifies the job to be queried (see job_id below)
common_args: Common arguments
job_id: If –1 is passed, the *object* argument identifies the URI of the job to be queried. Otherwise, *job_id* is the local job ID and the *object* argument identifies the URI of the printer the job is associated with.
requested_attrs: Array of attributes to be queried. If NULL is passed then all attributes are queried. (NOTE: The job may return more attributes than you requested. This is merely an advisory request that may reduce the amount of data returned if the printer/server supports it.)
num_requested_attrs: Number of requested attributes in the requested_attrs array. If zero is passed then all attributes are queried.

Returns: A pointer to a results structure containing error information (if any) and the attributes returned from the job object. If a NULL value is returned, a connection could not be made to the job object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include “papi.h”

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/jobs/1783”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiQueryJob(&object, &common_args, -1, NULL, 0);
```

See Also:
papiDelete, papiGetStatus, papiFindAttribute, papiGetNextAttribute

6.5. papiCancelJob

Description: Cancel the specified print job.

Syntax:

```
papi_result_t* papiCancelJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const int job_id
                        );
```

Arguments:

object: Identifies the printer or job (see job_id below)

common_args: Common arguments

job_id: If –1 is passed, the *object* argument identifies the URI of the job to be cancelled. Otherwise, *job_id* is the local job ID and the *object* argument identifies the URI of the printer the job is associated with.

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include “papi.h”

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/jobs/147”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiCancelJob(&object, &common_args, -1);
```

See Also:

papiDelete, papiGetStatus, papiGetErrorString

6.6. papiHoldJob

Description: Holds the specified print job and prevents it from being scheduled for printing. This operation is optional and may not be supported by all printers/servers. Use papiReleaseJob to undo the effects of this operation, or specify the hold_until argument to automatically release the job at a specific time.

See IPP's RFC 2911, section 3.3.5 "Hold-Job Operation" for further details on the semantics of this operation.

Syntax:

```
papi_result_t* papiHoldJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const int job_id,
            const char* hold_until
                        );
```

Arguments:

object: Identifies the printer or job (see job_id below)

common_args: Common arguments

job_id:

If –1 is passed, the *object* argument identifies the URI of the job to be held. Otherwise, *job_id* is the local job ID and the *object* argument identifies the URI of the printer the job is associated with.

hold_until:

(optional) Specifies the time when the job is to be automatically released for printing. If NULL, the job is held until explicitly released by calling papiReleaseJob. If specified, the value must be one of the strings "indefinite" (same effect as passing NULL), "day-time", "evening", "night", "weekend", "second-shift", or "third-shift". For values other than "indefinite", the printer/server must define exact times associated with these values and it may make these associations configurable.

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631/jobs/147");
```

```
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiHoldJob(&object, &common_args, -1, NULL);
```

See Also:
papiDelete, papiGetStatus, papiGetErrorString, papiReleaseJob

6.7. papiReleaseJob

Description: Releases the specified print job, allowing it to be scheduled for printing. This operation is optional and may not be supported by all printers/servers, but it must be supported if papiHoldJob is supported.

See IPP's RFC 2911, section 3.3.6 "Release-Job Operation" for further details on the semantics of this operation.

Syntax:

```
papi_result_t* papiReleaseJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const int job_id
                              );
```

Arguments:

object: Identifies the printer or job (see job_id below)

common_args: Common arguments

job_id:

If –1 is passed, the *object* argument identifies the URI of the job to be released. Otherwise, *job_id* is the local job ID and the *object* argument identifies the URI of the printer the job is associated with.

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include "papi.h"

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631/jobs/147");
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiReleaseJob(&object, &common_args, -1);
```

See Also:

papiDelete, papiGetStatus, papiGetErrorString, papiHoldJob

6.8. papiRestartJob

Description: Restarts a job that was retained after processing. If and how a job is retained after processing is implementation-specific and is not covered by this API. This operation is optional and may not be supported by all printers/servers.

Syntax:

```
papi_result_t* papiRestartJob(
            const papi_object_id_t* object,
            const papi_common_args_t* common_args,
            const int job_id
                              );
```

Arguments:

object: Identifies the printer or job (see job_id below)

common_args: Common arguments

job_id:

If –1 is passed, the *object* argument identifies the URI of the job to be restarted. Otherwise, *job_id* is the local job ID and the *object* argument identifies the URI of the printer the job is associated with.

Returns: A pointer to a results structure containing error information (if any). If a NULL value is returned, a connection could not be made to the printer object (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include “papi.h”

papi_object_id_t  object;
papi_common_args_t common_args;
papi_result_t* result;
…
strcpy(object.hostname, “printserv”);
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, “http://printserv:631/jobs/147”);
object.http = NULL;
PAPI_INIT_COMMON_ARGS(common_args);

result = papiRestartJob(&object, &common_args, -1);
```

See Also:

papiDelete, papiGetStatus, papiGetErrorString

7. Other API

7.1. papiDelete

Description: Delete the specified results.

Syntax:

```
void papiDelete(papi_result_t* result);
```

Arguments:
result: Pointer to result structure to delete

Returns: Nothing

Example:

```
#include “papi.h”

papi_result_t* result;
…
/* some operation returns result */
…
/* done processing result */
papiDelete(result);
```

See Also:
papiQueryPrinter, papiSubmitJob

7.2. papiGetStatus

Description: Retrieves the status code from the specified results.

Syntax:

```
papi_status_t papiGetStatus(const papi_result_t* result);
```

Arguments:
result: Pointer to result structure from which to retrieve the status

Returns: A status code value. If a NULL result is passed in, the last error status recorded by the library is returned.

Example:

```
#include “papi.h”

papi_status_t  status;
…
/* some operation performed */

if ((status = papiGetStatus(result)) != PAPI_OK)
{
      /* Handle the error */
      …
}
```

See Also:
papiGetErrorString

7.3. papiGetErrorString

Description: Returns an error string associated with the specified results.

Syntax:

```
const char* papiGetErrorString(
                  const papi_result_t* result
                              );
```

Arguments:
result: Pointer to result structure from which to generate the message

Returns: A pointer to a message string describing the error. If there is no error associated with the specified result, NULL is returned.

Example:

```
#include “papi.h”

papi_status_t  status;
…
/* some operation performed */

if ((status = papiGetStatus(result)) != PAPI_OK)
{
      /* Handle the error */
      printf(“ERROR: %s\n”, papiGetErrorString(result));
}
```

See Also:
papiGetStatus

7.4. papiOpenConnection

Description: Open a connection to the printer/server that will be used across multiple API calls. When done, call papiCloseConnection to close the connection.

By default (when the "http" field of the object ID is NULL), the connection to the printer/server is opened and closed within each API call. papiOpenConnection and papiCloseConnection are only used as a performance enhancement if the calling application will make many API calls which access the printer/server in rapid succession.

Syntax:

```
papi_status_t papiOpenConnection(papi_object_id_t* object);
```

Arguments:
object: Identifies the printer/server to connect to

Returns: A status code indicating whether or not a connection was successfully opened. If something other than PAPI_OK is returned, a connection could not be made to the printer/server (e.g. incorrect URI, server/printer off or offline, etc.).

Example:

```
#include "papi.h"

papi_status_t  status;
papi_object_id_t  object;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631");
object.http = NULL;
…

if ((status = papiOpenConnection(&object)) != PAPI_OK)
{
      /* Handle the error */
      …
}

/* perform multiple API calls here */
…
papiCloseConnection(&object);
```

See Also:
papiCloseConnection

7.5. papiCloseConnection

Description: Close a connection to the printer/server that was previously opened with papiOpenConnection.

By default (when the "http" field of the object ID is NULL), the connection to the printer/server is opened and closed within each API call. papiOpenConnection and papiCloseConnection are only used as a performance enhancement if the calling application will make many API calls which access the printer/server in rapid succession.

Syntax:

```
void papiCloseConnection(papi_object_id_t* object);
```

Arguments:
object: Identifies the printer/server to close the connection to

Returns: none

Example:

```
#include "papi.h"

papi_status_t  status;
papi_object_id_t  object;
…
strcpy(object.hostname, "printserv");
object.port = 631;
object.encryption = HTTP_ENCRYPT_NEVER;
strcpy(object.uri, "http://printserv:631");
object.http = NULL;
…

if ((status = papiOpenConnection(&object)) != PAPI_OK)
{
      /* Handle the error */
      …
}

/* perform multiple API calls here */
…
papiCloseConnection(&object);
```

See Also:
papiOpenConnection

8. New Attributes

This section documents new attributes which are extensions to the IPP 1.1 standard. The attributes should be added to newer versions of the standard, but are not yet in the IPP documentation so they are described here.

[to be added]

9. Appendices

9.1. Appendix A: Change History

Version 0.2 (April 17, 2002):

- Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)
- Filled in "2.3.3. Encryption" and added information about encryption in "3.1. Object Identification"
- Added "short_name" field in "3.1. Object Identification"
- Added "3.8. Job Ticket (papi_job_ticket_t)"
- Added "4.4. papiPausePrinter"
- Added "4.5. papiResumePrinter"
- Added "4.6. papiPurgeJobs"
- Added optional job_ticket argument to "6.1. papiSubmitJob"
- Added optional passing of filenames by URI to "6.1. papiSubmitJob"
- Added "6.6. papiHoldJob"
- Added "6.7. papiReleaseJob"
- Added "6.8. papiRestartJob"

Version 0.1 (April 3, 2002):

- Original draft version

End of Document