Open Standard Print API (PAPI)

Version 1.0

Alan Hlava
IBM Printing Systems Division

Norm Jacobs
Sun Microsystems, Inc.

Michael R. Sweet
Easy Software Products

Open Standard Print API (PAPI): Version 1.0
by Alan Hlava, Norm Jacobs, and Michael R. Sweet

Version 1.0 Edition

Table of Contents

22

Chapter 1: Introduction

This document describes the Open Standard Print Application Programming Interface (API), also known as the "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 can 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:

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

Subsequent versions of this document will incorporate support for a Document object, notification, and additional functions and attributes to more completely align with the PWG semantic model.

Chapter 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 possible objects a printing system model might include are:

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

2.2 Model

The model on which the Open Standard Print API is derived from reflect the semantics defined by the Internet Printing 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 [RFC2911], Chapter 2. Additional IPP-related documents can be found in theReferences appendix

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

2.2.1 Print Service

Note that an implementation of the PAPI interface may use protocols other than IPP for communicating with a print service. The only requirement is that the implementation accept and return the data structures as defined in this document.

2.2.2 Printer

Printer objects are the target of print job requests. A printer object may represent an actual printer (if the printer itself supports PAPI), 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 by one or more names which may be short, local names (such as "prtr1") or longer global names (such as a URI like "http://printserv.mycompany.com:631/printers/prtr1", “ipp://printserv/printers/prt1”, “lpd://server/queue”, etc.). The PAPI implementation may detect and map short names to long global names in an implementation-specific manner.

2.2.3 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. They also contain (logically) the print data itself in the form of one or more "documents".

Job objects are identified by an integer "job ID" that is assumed to be unique within the scope of the printer object to which the job was submitted. Thus, the combination of printer name or URI and the integer job ID globally identify a job.

2.2.4 Document

Document objects are sub-units of a job object. Conceptually, they may each contain a separate set of attributes describing the document and specifying how it will be printed. They also contain (logically) the print data itself.

This version of PAPI does NOT support separate document objects, but they will be added in a future version. It is likely that this will be done by adding new "Open job", "Add document", and "Close job" functions to allow submitting a multiple document job and specifying separate attributes for each document.

2.3 Security

The security model of this API is based on the IPP security model, which uses HTTP security mechanisms as well as implementation-defined security policies.

2.3.1 Authentication

Authentication will be done by using methods appropriate to the underlying server/printer being used. For example, if the underlying printer/server is using IPP protocol then either HTTP Basic or HTTP Digest authentication might be used.

Authentication is supported by supplying a user name and password. If the user name and password are not passed on the API call, the call may fail with an error code indicating an authentication problem.

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 print service may be desirable in some environments. See the "encryption" field in the service object for information on how to request encryption on a print operation. Note that some print services may not support encryption. To comply with this standard, only the PAPI_ENCRYPT_NEVER value must be supported.

2.4 Globalization

The PAPI interface follows the conventions for globalization and translation of human-readable strings that are outlined in the IPP standards. A quick summary:

- Attribute names are never translated.
- Most text values are not translated.
- Supporting translation by PAPI implementation is optional.
- If translation is supported, only the values of the following attributes are translated: job-state-message, document-state-message, and printer-state-message.

The above is just a summary. For details, see [RFC2911] section 3.1.4 and [PWGSemMod] section 6.

Chapter 3: Common Structures

3.1 Conventions

- All "char *" variables and fields are pointers to standard C/C++ NULL-terminated strings. It is assumed that these strings are all UTF-8 encoded characters strings.
- All pointer arrays (e.g. "char **") are assumed to be terminated by NULL pointers. That is, the valid elements of the array are followed by an element containing a NULL pointer that marks the end of the list.

3.2 Service Object (papi_service_t)

This opaque structure is used as a "handle" to maintain information about the print service being used to handle the PAPI requests. It is typically created once, used on one or more subsequent PAPI calls, and then destroyed.

```
typedef void *papi_service_t;
```

Included in the information associated with a papi_service_t is a definition about how requests will be encrypted during communication with the print service.

```
typedef enum {
        PAPI_ENCRYPT_IF_REQUESTED,/* Encrypt if requested (TLS upgrade) */
        PAPI_ENCRYPT_NEVER              /* Never encrypt */
        PAPI_ENCRYPT_REQUIRED,          /* Encryption is required (TLS upgrade) */
        PAPI_ENCRYPT_ALWAYS             /* Always encrypt (SSL) */
} papi_encryption_t;
```

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

3.3 Attributes and Values (papi_attribute_t)

These are the structures defining how attributes and values are passed to and from PAPI.

```
/* Attribute Type */
typedef enum {
        PAPI_STRING,
        PAPI_INTEGER,
        PAPI_BOOLEAN,
        PAPI_RANGE,
        PAPI_RESOLUTION,
        PAPI_DATETIME,
        PAPI_COLLECTION
        PAPI_METADATA
} papi_attribute_value_type_t;

/* Resolution units */
typedef enum {
        PAPI_RES_PER_INCH = 3,
        PAPI_RES_PER_CM
} papi_res_t; /* Boolean values */

enum {
        PAPI_FALSE = 0,
        PAPI_TRUE = 1
};

typedef enum {
        PAPI_UNSUPPORTED = 0x10,
        PAPI_DEFAULT = 0x11,
        PAPI_UNKNOWN,
        PAPI_NO_VALUE,
        PAPI_NOT_SETTABLE = 0x15,
        PAPI_DELETE = 0x16
} papi_metadata_t;

struct papi_attribute_str;

/* Attribute Value */
typedef union {
        char *string;                   /* PAPI_STRING value */
        int integer;                    /* PAPI_INTEGER value */
        char boolean;                   /* PAPI_BOOLEAN value */
        struct {                        /* PAPI_RANGE value*/
                int lower;
                int upper;
        } range;
        struct {                        /* PAPI_RESOLUTION value*/
                int xres;
                int yres;
                papi_res_t units;
        } resolution
        time_t datetime;                /* PAPI_DATETIME value */
        struct papi_attribute_str **
                collection;             /* PAPI_COLLECTION value */
        papi_metadata_t metadata;
```

```
} papi_attribute_value_t;

/* Attribute and Values */
typedef struct papi_attribute_str {
        char *name;                              /* attribute name */
        papi_attribute_value_type_t type;        /* type of values */
        papi_attribute_value_t **values;         /* list of values */
} papi_attribute_t;
```

The following constants are used by the papiAttributeListAdd* functions to control how values are added to the list.

```
/* Attribute add flags (add_flags) */
#define PAPI_ATTR_APPEND      0x0001       /* Add values to attribute*/
#define PAPI_ATTR_REPLACE     0x0002       /* Delete existing values, then add */
#define PAPI_ATTR_EXCL        0x0004       /* Fail if attribute exists */
```

For the valid attribute names which may be supported, see The Attributes appendix.

3.4 Job Object (papi_job_t)

This opaque structure is used as a "handle" to information associated with a job object. This handle is returned in response to successful job creation, modification, query, or list operations. See the "papiJobGet*" functions to see what information can be retrieved from the job object using the handle.

3.5 Stream Object (papi_stream_t)

This opaque structure is used as a "handle" to a stream of data. See the "papiJobStream*" functions for further details on how it is used.

3.6 Printer Object (papi_printer_t)

This opaque structure is used as a "handle" to information associated with a printer object. This handle is returned in response to successful printer modification, query, or list operations. See the "papiPrinterGet*" functions to see what information can be retrieved from the printer object using the handle.

3.7 Job Ticket (papi_job_ticket_t)

This structure is 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 http://www.cip4.org/.

```
/* Job Ticket Format */
typedef enum {
        PAPI_JT_FORMAT_JDF = 0,         /* Job Definition Format */
        PAPI_JT_FORMAT_PWG = 1          /* PWG Job Ticket Format */
} papi_jt_format_t;

/* Job Ticket */
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,
                                                file_name must be specified */
        char *file_name;                /* Name of the file containing the job ticket data.
                                                If ticket_data is specified, then file_name
                                                is ignored. */
} papi_job_ticket_t;
```

The file_name field may contain absolute path names, relative path names or URIs ([RFC1738], [RFC2396]). In the event that the name contains an absolute or relative path name (relative to the current directory), the implementation MUST copy the file contents before returning. If the name contains a URI, the implementation SHOULD NOT copy the referenced data unless (or until) it is no longer feasible to maintain the reference. Feasibility limitations may arise out of security issues, name space issues, and/or protocol or printer limitations.

3.8 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_BAD_ARGUMENT,
        PAPI_JOB_TICKET_NOT_SUPPORTED
} papi_status_t;
```

NOTE: If a Particular implementation of PAPI does not support a requested function, PAPI_OPERATION_NOT_SUPPORTED must be returned from that function.

See [RFC2911], section 13.1 for further explanations of the meanings of these status values.

3.9 List Filter (papi_filter_t)

This structure is used to filter the objects that get returned on a list request. When many objects could be returned from the request, reducing the list using a filter may have significant performance and network traffic benefits.

```
typedef enum {
        PAPI_FILTER_BITMASK = 0
        /* future filter types may be added here */
} papi_filter_type_t;

typedef struct {
        papi_filter_type_t   type; /* Type of filter specified */
        union {
                /* Bitmask filter */
                struct {
                        unsigned int  mask; /* bit mask  */
                        unsigned int  value; /* bit value */
                } bitmask;
        /* future filter types may be added here */
        } filter;
} papi_filter_t;
```

For papiPrintersList requests, the following values may be OR-ed together and used in the papi_filter_t mask and value fields to limit the printers returned. The logic used is to select printers which satisfy: "(printer-type & mask) == (value & mask)". This allows for simple "positive logic" (checking for the presence of characteristics) when mask and value are identical, and it also allows for "negative logic" (checking for the absence of characteristics) when they are different. For example, to select local (i.e. NOT remote) printers that support color:

```
papi_filter_t filter; filter.type = PAPI_FILTER_BITMASK;
filter.filter.bitmask.mask  = PAPI_PRINTER_REMOTE | PAPI_PRINTER_COLOR;
filter.filter.bitmask.value = PAPI_PRINTER_COLOR;
```

The filter bitmask values are:

```
enum {
        PAPI_PRINTER_LOCAL = 0x0000,          /* Local printer or class */
        PAPI_PRINTER_CLASS = 0x0001,          /* Printer class */
        PAPI_PRINTER_REMOTE = 0x0002,         /* Remote printer or class */
        PAPI_PRINTER_BW = 0x0004,             /* Can do B&W printing */
        PAPI_PRINTER_COLOR = 0x0008,          /* Can do color printing */
        PAPI_PRINTER_DUPLEX = 0x0010,         /* Can do duplexing */
        PAPI_PRINTER_STAPLE = 0x0020,         /* Can staple output */
        PAPI_PRINTER_COPIES = 0x0040,         /* Can do copies */
        PAPI_PRINTER_COLLATE = 0x0080,        /* Can collage copies */
        PAPI_PRINTER_PUNCH = 0x0100,          /* Can punch output */
        PAPI_PRINTER_COVER = 0x0200,          /* Can cover output */
        PAPI_PRINTER_BIND = 0x0400,           /* Can bind output */
        PAPI_PRINTER_SORT = 0x0800,           /* Can sort output */
        PAPI_PRINTER_SMALL = 0x1000,        /* Can do Letter/Legal/A4 */
        PAPI_PRINTER_MEDIUM = 0x2000,       /* Can do Tabloid/B/C/A3/A2 */
        PAPI_PRINTER_LARGE = 0x4000,        /* Can do D/E/A1/A0 */
        PAPI_PRINTER_VARIABLE = 0x8000,     /* Can do variable sizes */
        PAPI_PRINTER_IMPLICIT = 0x10000,    /* Implicit class */
        PAPI_PRINTER_DEFAULT = 0x20000,     /* Default printer on network */
        PAPI_PRINTER_OPTIONS = 0xfffc     /* ~(CLASS | REMOTE | IMPLICIT) */
};
```

3.10 Encryption (papi_encrypt_t)

This enumeration is used to get/set the encryption type to be used during communication with the print service.

```
typedef enum {
        PAPI_ENCRYPT_IF_REQUESTED,
        PAPI_ENCRYPT_NEVER,
        PAPI_ENCRYPT_REQUIRED,
        PAPI_ENCRYPT_ALWAYS
} papi_encryption_t;
```

Chapter 4: Attributes API

The interface described in this section is central to the PAPI printing model. Virtually all of the operations that can be performed against the print service objects (via function calls) make use of attributes. Object creation or modification operations tend to take in attribute list dewscribing the object or the requested modifications. Object creation, modification, queyr and list operations tent to return updated lists of print service objects containing attribute lists to more completely describe the objects.

In the case of a printer object, it's associated attribute list can be retreived using papiPrinterGetAttributeList. Job object attribute lists can be retreived using papiJobGetAttributeList. Once retreived, these attribute lists can be searched (or enumerated) to gather further information about the associated object. When creating or modifying print service objects, attribute lists can be built and passed into the create/modify operation. As a general rule of thumb, application developers should not modify or destroy attribute lists that they did not create. Modification or descruction of attribute lists retreived from print service objects should be handled by the PAPI implementation upon object destruction (free).

Because the attribute interface has specific functions to easy the use of various types of data that can be contained in an attribute list, there are a few things that are common to all of the papiAttributeAdd* fuctions and some common to all of the papiAttribute ListGet* functions.

All of the papiAttributeListAdd* functions take in a pointer to an attribute list, a set of flags, an attribute name, and call/type specific values. For all of the papiAttributeListAdd* functions, the attribute list pointer (papi_attribute_t ***attrs) may not contain a NULL value. If a NULL value is passed to any of these functions, the function must return PAPI_BAD_ARGUMENT. The flags passed into each of the papiAttributeListAdd* calls describe how the attribute/values are to be added to the attribute list. Currently, there are three flags that can be passed: PAPI_ATTR_EXCL, PAPI_ATTR_REPLACE, and PAPI_ATTR_APPEND. If PAPI_ATTR_EXCL is passed, it indicates that this call should only succeed if the named attribute does not already exist in the attribute list. PAPI_ATTR_REPLACE indicates that prior to addition to the attribute list, this call should truncate any existing attribute values for the named attribute if is already contained in the list. PAPI_ATTR_APPEND indicates that any attribute values contained in this call should be appended to the named attribute's value list if the named attribute was already contained in the attribute list.

All of the papiAttributeListGet* functions take in an attribute list, iterator, name, and pointer(s) for type specific results. If the named attribute is found in the attirbute list, but it's type does not match the type supplied in papiAttributeListGet or the type implied by the various type specific calls, a value of PAPI_NOT_POSSIBLE must be returned from the call. Any papiAttributeListGet* failure must not modify the information in the provided results arguments.

4.1 papiAttributeListAdd

4.1.1 Description

Add an attribute/value to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.
This function is equivalent to the papiAttributeListAddString, papiAttributeListAddInteger, papiAttributeListAddBoolean, papiAttributeListAddRange, papiAttributeListAddResolution, papiAttributeListAddDatetime, papiAttributeListAddCollection, and papiAttributeListAddMetadata functions defined later in this chapter.

4.1.2 Syntax

```
papi_status_t papiAttributeListAdd(papi_attribute_t ***attrs, int add_flags,
                                   char *name, papi_attribute_value_type_t type,
                                   papi_attribute_value_t *value );
```

4.1.3 Inputs

4.1.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.1.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.1.3.3 name

Points to the name of the attribute to add.

4.1.3.4 type

The type of values for this attribute.

4.1.3.5 value

Points to the attribute value to be added.

4.1.4 Outputs

4.1.4.1 attrs

The attribute list is updated.

4.1.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.1.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
papi_attribute_value_t value;
...
value.string = “My Job”;
status = papiAttributeListAdd(&attrs, PAPI_ATTR_EXCL, "job-name",
                              PAPI_STRING, &value);
...
papiAttributeListFree(attrs);
```

4.1.7 See Also

papiAttributeListAddString, papiAttributeListAddInteger, papiAttributeListAddBoolean, papiAttributeListAddRange, papiAttributeListAddResolution, papiAttributeListAddDatetime, papiAttributeListAddCollection, papiAttributeListFromString, papiAttributeListFree

4.2 papiAttributeListAddString

4.2.1 Description

Add a string-valued attribute to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.2.2 Syntax

```
papi_status_t papiAttributeListAddString(papi_attribute_t ***attrs, int add_flags,
                                         char *name, char *value);
```

4.2.3 Inputs

4.2.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.2.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.2.3.3 name

Points to the name of the attribute to add.

4.2.3.4 value

The string value to be added to the attribute.

4.2.4 Outputs

4.2.4.1 attrs

The attribute list is updated.

4.2.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.2.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
...
status = papiAttributeListAddString(&attrs, PAPI_ATTR_EXCL,
                                   "job-name", "My job" );
...
papiAttributeListFree(attrs);
```

4.2.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.3 papiAttributeListAddInteger

4.3.1 Description

Add an integer-valued attribute to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.3.2 Syntax

```
papi_status_t papiAttributeListAddInteger(papi_attribute_t ***attrs, int add_flags,
                                char *name,int value );
```

4.3.3 Inputs

4.3.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.3.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.3.3.3 name

Points to the name of the attribute to add.

4.3.3.4 value

The integer value to be added to the attribute.

4.3.4 Outputs

4.3.4.1 attrs

The attribute list is updated.

4.3.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.3.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
...
status = papiAttributeListAddInteger(&attrs, PAPI_ATTR_EXCL,
                       "copies", 3);
...
papiAttributeListFree(attrs);
```

4.3.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.4 papiAttributeListAddBoolean

4.4.1 Description

Add a boolean-valued attribute to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.4.2 Syntax

```
papi_status_t papiAttributeListAddBoolean(papi_attribute_t ***attrs, int add_flags,
                            char *name, char value );
```

4.4.3 Inputs

4.4.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.4.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.4.3.3 name

Points to the name of the attribute to add.

4.4.3.4 value

The boolean value (PAPI_FALSE or PAPI_TRUE) to be added to the attribute.

4.4.4 Outputs

4.4.4.1 attrs

The attribute list is updated.

4.4.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.4.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
```

```
...
status = papiAttributeListAddBoolean(&attrs, PAPI_ATTR_EXCL,
                         ”color-supported”, PAPI_TRUE);
...
papiAttributeListFree(attrs);
```

4.4.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.5 papiAttributeListAddRange

4.5.1 Description

Add a range-valued attribute to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.5.2 Syntax

```
papi_status_t papiAttributeListAddRange(papi_attribute_t ***attrs, int add_flags,
                              char *name, int lower, int upper);
```

4.5.3 Inputs

4.5.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.5.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.5.3.3 name

Points to the name of the attribute to add.

4.5.3.4 lower

An integer value representing the lower boundary of a range value. This value must be less than or equal to the upper range value.

4.5.3.5 upper

An integer value representing the upper boundary of the range value. This value must be greater than or equal to the lower range value

4.5.4 Outputs

4.5.4.1 attrs

The attribute list is updated.

4.5.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.5.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
...
status = papiAttributeListAddRange(&attrs, PAPI_ATTR_EXCL,
                         "job-k-octets-supported",1, 100000);
...
papiAttributeListFree(attrs);
```

4.5.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.6 papiAttributeListAddResolution

4.6.1 Description

Add a resolution-valued attribute to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.6.2 Syntax

```
papi_status_t papiAttributeListAddResolution(papi_attribute_t ***attrs,
                              int add_flags, char *name,
                              int xres, int yres, papi_res_t units);
```

4.6.3 Inputs

4.6.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.6.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.6.3.3 name

Points to the name of the attribute to add.

4.6.3.4 xres

The integer X-axis resolution value.

4.6.3.5 yres

The integer Y-axis resolution value.

4.6.3.6 Units

The units of the X-axis and y-axis resolution values provided.

4.6.4 Outputs

4.6.4.1 attrs

The attribute list is updated.

4.6.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.6.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
...
status = papiAttributeListAddResolution(&attrs, PAPI_ATTR_EXCL,
                        “printer-resolution”, 300, 300,
                        PAPI_RES_PER_INCH);
...
papiAttributeListFree(attrs);
```

4.6.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.7 papiAttributeListAddDatetime

4.7.1 Description

Add a date/time-valued attribute to an attribute list. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.7.2 Syntax

```
papi_status_t papiAttributeListAddDatetime(papi_attribute_t ***attrs, int add_flags,
                                 char *name, time_t value );
```

4.7.3 Inputs

4.7.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.7.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.7.3.3 name

Points to the name of the attribute to add.

4.7.3.4 value

The time_t representation of the date/time value to be added to the attribute.

4.7.4 Outputs

4.7.4.1 attrs

The attribute list is updated.

4.7.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.7.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
```

```
time_t date_time;
...
time(&date_time);
status = papiAttributeListAdd(&attrs, PAPI_EXCL,
                  "date-time-at-creation", date_time);
...
papiAttributeListFree(attrs);
```

4.7.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.8 papiAttributeListAddCollection

4.8.1 Description

Add a collection-valued attribute to an attribute list. A collection-valued attribute is a container for list of attributes. Depending on the add_flags, this may also be used to add values to an existing multi-valued attribute. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.8.2 Syntax

```
papi_status_t papiAttributeListAddCollection(papi_attribute_t ***attrs,
                  int add_flags, char *name,
                  papi_attribute_t **collection);
```

4.8.3 Inputs

4.8.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.8.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.8.3.3 name

Points to the name of the attribute to add.

4.8.3.4 collection

Points to the attribute list to be added as a collection.

4.8.4 Outputs

4.8.4.1 attrs

The attribute list is updated.

4.8.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.8.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
papi_attribute_t **collection = NULL;
...
/* create the collection /
status = papiAttributeListAddString(&collection, PAPI_EXCL,
                        "media-key", “iso-a4-white”);
status = papiAttributeListAddString(&collection, PAPI_EXCL,
                        "media-type", “stationery”);
...
/ add the collection to the attribute list */
status = papiAttributeListAddCollection(&attrs, PAPI_EXCL,
                        "media-col", collection);
...
papiAttributeListFree(collection);
papiAttributeListFree(attrs);
```

4.8.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.9 papiAttributeListAddMetadata

4.9.1 Description

Add a meta-valued attribute to an attribute list. A meta-valued attribute is a container for attribute information not normally represented in an attribute value. Memory is allocated and copies of the input arguments are created. It is the caller's responsibility to call papiAttributeListFree when done with the attribute list.

4.9.2 Syntax

```
papi_status_t papiAttributeListAddMetadata(papi_attribute_t ***attrs,
                        int add_flags, char *name,
                        papi_metadata_t value);
```

4.9.3 Inputs

4.9.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.9.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.9.3.3 name

Points to the name of the attribute to add.

4.9.3.4 value

The type of metadata to be added to the attribute. PAPI_DELETE can be used to indicate that an attribute should be removed from a print service object when calling one of the papi*Modify functions. PAPI_DEFAULT can be used to indicate that the print service should set (or reset) the named attribute value to a "default" value during a create or modify operation of a print service object.

4.9.4 Outputs

4.9.4.1 attrs

The attribute list is updated.

4.9.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.9.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
...
/ add the collection to the attribute list */
status = papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
                         "media", PAPI_DELETE);
...
papiAttributeListFree(collection);
papiAttributeListFree(attrs);
```

4.9.7 See Also

papiAttributeListAdd, papiAttributeListFree

4.10 papiAttributeListDelete

4.10.1 Description

Delete an attribute from an attribute list. All memory associated with the deleted attribute is deallocated.

4.10.2 Syntax

```
papi_status_t papiAttributeListDelete(papi_attribute_t ***attrs, char *name);
```

4.10.3 Inputs

4.10.3.1 attrs

Points to an attribute list.

4.10.3.2 name

Points to the name of the attribute to remove.

4.10.4 Outputs

4.10.4.1 attrs

The attribute list is updated.

4.10.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.10.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
...
status = papiAttributeListAddDelete(&attrs, “copies”);
...
papiAttributeListFree(attrs);
```

4.10.7 See Also

papiAttributeListFree

4.11 papiAttributeListGetValue

4.11.1 Description

Get an attribute's value from an attribute list.
This function is equivalent to the papiAttributeListGetString, papiAttributeListGetInteger, papiAttributeListGetBoolean, papiAttributeListGetRange, papiAttributeListGetResolution, papiAttributeListGetDatetime, and papiAttributeListGetCollection functions defined later in this chapter.

4.11.2 Syntax

```
papi_status_t papiAttributeListGetValue(papi_attribute_t **attrs, void **iterator,
                                        char *name, papi_attribute_value_type_t type,
                                        papi_attribute_t **value);
```

4.11.3 Inputs

4.11.3.1 attrs

Points to an attribute list.

4.11.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.11.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be located in the attribute list supplied, PAPI_NOT_FOUND is returned.

4.11.3.4 type

The type of values for this attribute. If the type supplied does not match the type of the named attribute in the attribute list, PAPI_NOT_POSSIBLE is returned.

4.11.4 Outputs

4.11.4.1 Iterator

See iterator in the Inputs section above

4.11.4.2 value

Points to the variable where a pointer to the attribute value is to be returned. Note that the

returned pointer points to the attribute's value in the list (no copy of the value is made) so that the caller does not need to do any special cleanup of the returned value's memory (it is cleaned up when the containing attribute list is deallocated).
If this call returns an error, the output value is not changed.

4.11.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.11.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
papi_attribute_value_t *job_name_value;
...
status = papiAttributeListGetValue(attrs, NULL, “job-name”,
                        PAPI_STRING, &job_name_value);
...
papiAttributeListFree(attrs);
```

4.11.7 See Also

papiAttributeListGetString, papiAttributeListGetInteger, papiAttributeListGetBoolean, papiAttributeListGetRange, papiAttributeListGetResolution, papiAttributeListGetDatetime, papiAttributeListGetCollection, papiAttributeListFree

4.12 papiAttributeListGetString

4.12.1 Description

Get a string-valued attribute's value from an attribute list.

4.12.2 Syntax

```
papi_status_t papiAttributeListGetString(papi_attribute_t **attrs, void **iterator,
                        char *name, char **value);
```

4.12.3 Inputs

4.12.3.1 attrs

Points to an attribute list.

4.12.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.12.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_STRING, PAPI_NOT_POSSIBLE will be returned.

4.12.4 Outputs

4.12.4.1 Iterator

See iterator in the Inputs section above

4.12.4.2 value

Pointer to the string (char *) where a pointer to the value is returned. If this call returns an error, the output value is not changed. Note that the returned pointer points to the attribute's value in the list (no copy of the value is made) so that the caller does not need to perform any special cleanup of the returned value's memory (it is cleaned up when the containing attribute list is deallocated).

4.12.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.12.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
char *value = NULL;
...
status = papiAttributeListGetString(attrs, NULL, “job-name”,
                        PAPI_STRING, &value);
...
papiAttributeListFree(attrs);
```

4.12.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.13 papiAttributeListGetInteger

4.13.1 Description

Get an integer-valued attribute's value from an attribute list.

4.13.2 Syntax

```
papi_status_t papiAttributeListGetInteger(papi_attribute_t **attrs, void **iterator,
                              char *name, int *value);
```

4.13.3 Inputs

4.13.3.1 attrs

Points to an attribute list.

4.13.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.13.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_INTEGER, PAPI_NOT_POSSIBLE will be returned.

4.13.4 Outputs

4.13.4.1 Iterator

See iterator in the Inputs section above

4.13.4.2 value

Pointer to the int where the value is returned. The value from the attribute list is copied to this location. If this call returns an error, the output value is not changed.

4.13.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.13.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
int value = 0;
...
status = papiAttributeListGetInteger(attrs, NULL, “copies”, &value);
...
papiAttributeListFree(attrs);
```

4.13.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.14 papiAttributeListGetBoolean

4.14.1 Description

Get a boolean-valued attribute's value from an attribute list.

4.14.2 Syntax

```
papi_status_t papiAttributeListGetBoolean(papi_attribute_t **attrs, void **iterator,
                              char *name, char *value);
```

4.14.3 Inputs

4.14.3.1 attrs

Points to an attribute list.

4.14.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.14.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_BOOLEAN, PAPI_NOT_POSSIBLE will be returned.

4.14.4 Outputs

4.14.4.1 Iterator

See iterator in the Inputs section above.

4.14.4.2 value

Pointer to the char where the value is returned. The value from the attribute list is copied to this location. If this call returns an error, the output value is not changed.

4.14.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.14.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
char value = PAPI_FALSE;
...
status = papiAttributeListGetBoolean(attrs, NULL,
                   “color-supported”, &value);
...
papiAttributeListFree(attrs);
```

4.14.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.15 papiAttributeListGetRange

4.15.1 Description

Get a range-valued attribute's values from an attribute list.

4.15.2 Syntax

```
papi_status_t papiAttributeListGetRange(papi_attribute_t **attrs, void **iterator,
                                        char *name, int *lower, int *upper);
```

4.15.3 Inputs

4.15.3.1 attrs

Points to an attribute list.

4.15.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.15.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_RANGE, PAPI_NOT_POSSIBLE will be returned.

4.15.4 Outputs

4.15.4.1 Iterator

See iterator in the inputs section above.

4.15.4.2 lower

Pointer to the integer where the values are returned. The value from the attribute list is copied to this location. If this call returns an error, the output values are not changed.

4.15.4.3 upper

Pointer to the integer where the values are returned. The value from the attribute list is copied to this location. If this call returns an error, the output values are not changed.

4.15.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.15.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
int lower = 0;
int upper = 0;
...
status = papiAttributeListGetRange(attrs, NULL,
                  “job-k-octets-supported”, &lower, &upper);
...
papiAttributeListFree(attrs);
```

4.15.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.16 papiAttributeListGetResolution

4.16.1 Description

Get a resolution-valued attribute's value from an attribute list.

4.16.2 Syntax

```
papi_status_t papiAttributeListGetResolution(papi_attribute_t **attrs, void **iterator,
                                char *name, int *xres, int *yres, papi_res_t *units);
```

4.16.3 Inputs

4.16.3.1 attrs

Points to an attribute list.

4.16.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.16.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_RESOLUTION, PAPI_NOT_POSSIBLE will be returned.

4.16.4 Outputs

4.16.4.1 iterator

See iterator in the Inputs section above.

4.16.4.2 xres

Pointer to the int where the X-resolution value is returned. The value from the attribute list is copied to this location. If this call returns an error, the output value is not changed.

4.16.4.3 yres

Pointer to the int where the Y-resolution value is returned. The value from the attribute list is copied to this location. If this call returns an error, the output value is not changed.

4.16.4.4 units

Pointer to the variable where the resolution-units value is returned. The value from the attribute list is copied to this location. If this call returns an error, the output value is not changed.

4.16.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.16.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
int xres, yres;
papi_res_t units;
...
status = papiAttributeListGetResolution(attrs, NULL,
                  “printer-resolution”, &xres, &yres, &units);
...
papiAttributeListFree(attrs);
```

4.16.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.17 papiAttributeListGetDatetime

4.17.1 Description

Get a datetime-valued attribute's value from an attribute list.

4.17.2 Syntax

```
papi_status_t papiAttributeListGetDatetime(papi_attribute_t **attrs, void **iterator,
                        char *name, time_t *value);
```

4.17.3 Inputs

4.17.3.1 attrs

Points to an attribute list.

4.17.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.17.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_DATETIME, PAPI_NOT_POSSIBLE will be returned.

4.17.4 Outputs

4.17.4.1 iterator

See iterator in the Inputs section above

4.17.4.2 value

Pointer to the time_t where the value is returned. The value from the attribute list is copied to this location. If this call returns an error, the output value is not changed.

4.17.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.17.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
time_t value = 0;
...
status = papiAttributeListGetDatetime(attrs, NULL,
                 “date-time-at-creation”, &value);
...
papiAttributeListFree(attrs);
```

4.17.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.18 papiAttributeListGetCollection

4.18.1 Description

Get a collection-valued attribute's value from an attribute list.

4.18.2 Syntax

```
papi_status_t papiAttributeListGetCollection(papi_attribute_t **attrs, void **iterator,
                                char *name, papi_attribute_t ***value);
```

4.18.3 Inputs

4.18.3.1 attrs

Points to an attribute list.

4.18.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.18.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_COLLECTION, PAPI_NOT_POSSIBLE will be returned.

4.18.3.4 type

The type of values for this attribute.

4.18.4 Outputs

4.18.4.1 Iterator

See iterator in the Inputs section above.

4.18.4.2 value

Points to the variable where a pointer to the attribute value is to be returned. Note that the returned pointer points to the attribute's value in the list (no copy of the value is made) so that the caller does not need to do any special cleanup of the returned value's memory (it is cleaned up when the containing attribute list is deallocated).
If this call returns an error, the output value is not changed.

4.18.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.18.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
papi_attribute_t **value = NULL;
...
status = papiAttributeListGetCollection(attrs, NULL,
                         “media-col”, &value);
...
papiAttributeListFree(attrs);
```

4.18.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.19 papiAttributeListGetMetadata

4.19.1 Description

Get a meta-valued attribute's value from an attribute list.

4.19.2 Syntax

```
papi_status_t papiAttributeListGetMetadata(papi_attribute_t **attrs, void **iterator,
                            char *name, papi_metadata_t *value);
```

4.19.3 Inputs

4.19.3.1 attrs

Points to an attribute list.

4.19.3.2 iterator

(optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multi-valued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

4.19.3.3 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is found in the attribute list, but is not a PAPI_STRING, PAPI_NOT_POSSIBLE will be returned.

4.19.3.4 type

The type of values for this attribute.

4.19.4 Outputs

4.19.4.1 Iterator

See iterator in the Inputs section above.

4.19.4.2 value

Points to the variable where the attribute value is to be returned. If this call returns an error, the output value is not changed.

4.19.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.19.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
papi_metadata_t value = PAPI_NO_VALUE;
...
status = papiAttributeListGetMetadata(attrs, NULL,
                        “media”, &value);
...
papiAttributeListFree(attrs);
```

4.19.7 See Also

papiAttributeListGetValue, papiAttributeListFree

4.20 papiAttributeListFree

4.20.1 Description

Frees an attribute list

4.20.2 Syntax

```
void papiAttributeListFree(papi_attribute_t **attrs);
```

4.20.3 Inputs

4.20.3.1 attrs

Attribute list to be deallocated.

4.20.4 Outputs

none

4.20.5 Returns

none

4.20.6 Example

```
papi_attribute_t **attrs = NULL;
...
papiAttributeListFree(attrs);
```

4.20.7 See Also

papiAttributeListAdd, papiAttributeListAddString, papiAttributeListAddInteger, papiAttributeListAddBoolean, papiAttributeListAddRange, papiAttributeListAddResolution, papiAttributeListAddDatetime, papiAttributeListAddCollection, papiAttributeListFromString, papiAttributeListFree

4.21 papiAttributeListFind

4.21.1 Description

Find an attribute in an attribute list.

4.21.2 Syntax

```
papi_attribute_t *papiAttributeListFind(papi_attribute_t **attrs, char *name);
```

4.21.3 Inputs

4.21.3.1 attrs

Points to an attribute list.

4.21.3.2 name

Points to the name of the attribute to retrieve. If the named attribute can not be found in the supplied attribute list, PAPI_NOT_FOUND will be returned.

4.21.4 Outputs

none

4.21.5 Returns

Pointer to the named attribute found in the attribute list. The result will be deallocated when the containing attribute list is destroyed. NULL indicates that the specified attribute was not found

4.21.6 Example

```
papi_attribute_t **attrs = NULL;
papi_attribute_t *value;
...
value = papiAttributeListFind(attrs, “job-name”);
...
papiAttributeListFree(attrs);
```

4.21.7 See Also

papiAttributeListGetValue

4.22 papiAttributeListGetNext

4.22.1 Description

Get the next attribute in an attribute list.

4.22.2 Syntax

```
papi_attribute_t **papiAttributeListGetNext(papi_attribute_t **attrs, void **iterator);
```

4.22.3 Inputs

4.22.3.1 attrs

Points to an attribute list.

4.22.3.2 iterator

Pointer to an opaque (void*) iterator. This should be NULL to find the first attribute and then passed in unchanged on subsequent calls to this function.

4.22.4 Outputs

4.22.4.1 Iterator

See iterator in the Inputs section above.

4.22.5 Returns

Pointer to the next attribute in the attribute list. The result will be deallocated when the containing attribute list is destroyed. NULL indicates that the end of the attribute list was reached

4.22.6 Example

```
papi_attribute_t **attrs = NULL;
papi_attribute_t *value;
void *iterator = NULL;
...
while ((value = papiAttributeGetNext(attrs, &iterator)) != NULL) {
      ...
}
...
papiAttributeListFree(attrs);
```

4.22.7 See Also

papiAttributeListFind

4.23 papiAttributeListFromString

4.23.1 Description

Convert a string of text options to an attribute list. PAPI provides two functions which map job attributes to and from text options that are typically provided on the command-line by the user. This text encoding is also backwards-compatible with existing printing systems and is relatively simple to parse and generate. See Attribute List Text Representationfor a definition of the string syntax.

4.23.2 Syntax

```
papi_status_t papiAttributeListFromString(papi_attribute_t*** attrs,
                                int add_flags, char* buffer );
```

4.23.3 Inputs

4.23.3.1 attrs

Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

4.23.3.2 add_flags

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request.

4.23.3.3 buffer

Points to text options.

4.23.4 Outputs

4.23.4.1 attrs

The attribute list is updated.

4.23.5 Returns

If the text string is successfully converted to an attribute list, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.23.6 Example

```
papi_status_t status;
papi_attribute_t **attrs = NULL;
char *string = "copies=1 job-name=John\'s\ Really\040Nice\ Job";
...
status = papiAttributeListFromString(attrs, PAPI_ATTR_EXCL, string);
...
papiAttributeListFree(attrs);
```

4.23.7 See Also

papiAttributeListFind

4.24 papiAttributeListToString

4.24.1 Description

Convert an attribute list to its text representation. The destination string is limited to at most (buflen - 1) bytes plus the trailing null byte.
PAPI provides two functions which map job attributes to and from text options that are typically provided on the command-line by the user. This text encoding is also backwards-

compatible with existing printing systems and is relatively simple to parse and generate. See Attribute List Text Representation for a definition of the string syntax.

4.24.2 Syntax

```
papi_status_t papiAttributeListToString(papi_attribute_t** attrs,
                                        char* attr_delim, char* buffer,
                                        size_t buflen );
```

4.24.3 Inputs

4.24.3.1 attr

Points to an attribute list.

4.24.3.2 attr_delim

(optional) If not NULL, points to a string to be placed between attributes in the output buffer. If NULL, a space is used as the attribute delimiter.

4.24.3.3 buffer

Points to a string buffer to receive the to receive the text representation of the attribute list.

4.24.3.4 buflen

Specifies the length of the string buffer in bytes.

4.24.4 Outputs

4.24.4.1 buffer

The buffer is filled with the text representation of the attribute list. The buffer will always be set to something by this function (buffer[0] = NULL in cases of an error).

4.24.5 Returns

If the attribute list is successfully converted to a text string, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

4.24.6 Example

```
papi_attribute_t **attrs = NULL;
char buffer[8192];
...
papiAttributeListToString(attrs, NULL, buffer, sizeof (buffer));
...
papiAttributeListFree(attrs);
```

4.24.7 See Also

PapiAttributeListFromString

Chapter 5: Service API

The service segment of the PAPI provides a means of creating, modifying, or destroying a context (or object) used to interact with a print service. This context is opaque to applications using it and may be used by implementations to store internal data such as file or socket descriptors, operation results, credentials, etc.

5.1 papiServiceCreate

5.1.1 Description

Create a print service handle to be used in subsequent calls. Memory is allocated and copies of the input arguments are created so that the handle can be used outside the scope of the input variables.
The caller must call papiServiceDestroy when done in order to free the resources associated with the print service handle. This must be done even if the papiServiceCreate call failed, because a service creation failure may have resulted in a partial service context with additional error information.

5.1.2 Syntax

```
papi_status_t papiServiceCreate( papi_service_t *handle, char *service_name,
                                 char *user_name, char *password,
                                 int (*authCB)(papi_service_t svc),
                                 papi_encryption_t encryption, void *app_data );
```

5.1.3 Inputs

5.1.3.1 service_name

(optional) Points to the name or URI of the service to use. A NULL value indicates that a “default service” should be used (the configuration of a default service is implementation-specific and may consist of environment variables, config files, etc. Default service selection is not addressed by this standard).

5.1.3.2 user_name

(optional) Points to the name of the user who is making the requests. A NULL value indicates that the user name associated with the process in which the API call is made should be used.

5.1.3.3 Password

(optional) Points to the password to be used to authenticate the user to the print service.

5.1.3.4 AuthCB

(optional) Points to a callback function to be used in authenticating the user to the print service if no password was supplied (or user input is required). A NULL value indicates that no callback should be made. The callback function should return 0 if the request is to be canceled and non-zero if new authentication information has been set.

5.1.3.5 Encryption

Specifies the encryption type to be used by the PAPI functions.

5.1.3.6 app_data

(optional) Points to application-specific data for use by the callback. The caller is responsible for allocating and freeing memory associated with this data.

5.1.4 Outputs

5.1.4.1 handle

A print service handle to be used on subsequent API calls. The handle will always be set to something even if the function fails. In the event that the function fails, the handle may be set to NULL or it may be set to a valid handle that contains error information.

5.1.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

5.1.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                 “user”, “password”, NULL,
                 PAPI_ENCRYPT_IF_REQUESTED, NULL);
...
papiServiceDestroy(handle);
```

5.1.7 See Also

papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUserName, papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB, papiServiceSetAppData,.papiServiceGetStatusMessage

5.2 papiServiceDestroy

5.2.1 Description

Destroy a print service handle and free the resources associated with it. This must be called even if the papiServiceCreate call failed, because there may be error information associated with the returned handle. If there is application data associated with the service handle, it is the caller's responsibility to free this memory.

5.2.2 Syntax

```
void papiServiceDestroy(papi_service_t handle );
```

5.2.3 Inputs

5.2.3.1 handle

The print service handle to be destroyed.

5.2.4 Outputs

None

5.2.5 Returns

None

5.2.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                 “user”, “password”, NULL,
                 PAPI_ENCRYPT_IF_REQUESTED, NULL);
...
papiServiceDestroy(handle);
```

5.2.7 See Also

papiServiceCreate

5.3 papiServiceSetUserName

5.3.1 Description

Set the user name in the print service handle to be used in subsequent calls. Memory is allocated and a copy of the input argument is created so that the handle can be used outside the scope of the input variable.

5.3.2 Syntax

```
papi_status_t papiServiceSetUserName( papi_service_t handle,
                                      char* user_name );
```

5.3.3 Inputs

5.3.3.1 handle

Handle to the print service to update.

5.3.3.2 user_name

Points to the name of the user who is making the requests. A NULL value indicates that the user name associated with the process in which the API call is made should be used.

5.3.4 Outputs

5.3.4.1 handle

Handle remains unchanged, but it’s contents may be updated.

5.3.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

5.3.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                  “user”, “password”, NULL,
                  PAPI_ENCRYPT_IF_REQUESTED, NULL);
...
status = papiServiceSetUserName(handle, “root”);
...
papiServiceDestroy(handle);
```

5.3.7 See Also

papiServiceCreate, papiServiceGetUserName, papiServiceGetStatusMessage

5.4 papiServiceSetPassword

5.4.1 Description

Set the password in the print service handle to be used in subsequent calls. Memory is allocated and a copy of the input argument is created so that the handle can be used outside the scope of the input variable.

5.4.2 Syntax

```
papi_status_t papiServiceSetPassword( papi_service_t handle, char* password);
```

5.4.3 Inputs

5.4.3.1 handle

Handle to the print service to update.

5.4.3.2 password

Points to the password to be used to authenticate the user to the print service.

5.4.4 Outputs

5.4.4.1 handle

Handle remains unchanged, but it's contents may be updated.

5.4.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

5.4.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                  “user”, “password”, NULL,
                  PAPI_ENCRYPT_IF_REQUESTED, NULL);
...
status = papiServiceSetPassword(handle, “passsword”);
...
papiServiceDestroy(handle);
```

5.4.7 See Also

papiServiceCreate, papiServiceGetPassword, papiServiceGetStatusMessage

5.5 papiServiceSetEncryption

5.5.1 Description

Set the encryption in the print service handle to be used in subsequent calls.

5.5.2 Syntax

```
papi_status_t papiServiceSetEncryption( papi_service_t handle,
                                         papi_encryption_t encryption);
```

5.5.3 Inputs

5.5.3.1 handle

Handle to the print service to update.

5.5.3.2 encryption

Specifies the encryption type to be used by the PAPI functions.

5.5.4 Outputs

5.5.4.1 handle

Handle remains unchanged, but it's contents may be updated.

5.5.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

5.5.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                 “user”, “password”, NULL,
                 PAPI_ENCRYPT_IF_REQUESTED, NULL);
...
status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
...
papiServiceDestroy(handle);
```

5.5.7 See Also

papiServiceCreate, papiServiceGetEncryption, papiServiceGetStatusMessage

5.6 papiServiceSetAuthCB

5.6.1 Description

Set the authorization callback function in the print service handle to be used in subsequent calls.

5.6.2 Syntax

```
papi_status_t papiServiceSetAuthCB( papi_service_t handle,
                                    int (*authCB)(papi_service_t svc));
```

5.6.3 Inputs

5.6.3.1 handle

Handle to the print service to update.

5.6.3.2 authCB

Points to a callback function to be used in authenticating the user to the print service if no password was supplied (or user input is required). A NULL value indicates that no callback should be made. The callback function should return 0 if the request is to be canceled and non-zero if new authentication information has been set.

5.6.4 Outputs

5.6.4.1 handle

Handle remains unchanged, but it’s contents may be updated.

5.6.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

5.6.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                     “user”, “password”, NULL,
                     PAPI_ENCRYPT_IF_REQUESTED, NULL);
```

```
...
status = papiServiceSetAuthCB(handle, get_password_callback);
...
papiServiceDestroy(handle);
```

5.6.7 See Also

papiServiceCreate, papiServiceGetStatusMessage

5.7 papiServiceSetAppData

5.7.1 Description

Set a pointer to some application-specific data in the print service. This data may be used by the authentication callback function. The caller is responsible for allocating and freeing memory associated with this data.

5.7.2 Syntax

```
papi_status_t papiServiceSetAppData( papi_service_t handle, void *app_data);
```

5.7.3 Inputs

5.7.3.1 handle

Handle to the print service to update.

5.7.3.2 app_data

Points to application-specific data for use by the callback. The caller is responsible for allocating and freeing memory associated with this data.

5.7.4 Outputs

5.7.4.1 handle

Handle remains unchanged, but it's contents may be updated.

5.7.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

5.7.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
```

```
...
status = papiServiceCreate(&handle, “ipp://printserver:631”,
                  “user”, “password”, NULL,
                  PAPI_ENCRYPT_IF_REQUESTED, NULL);
...
status = papiServiceSetAppData(handle, app_data);
...
papiServiceDestroy(handle);
```

5.7.7 See Also

papiServiceCreate, papiServiceGetAppData, papiServiceGetStatusMessage

5.8 papiServiceGetServiceName

5.8.1 Description

Get the service name associated with the print service handle.

5.8.2 Syntax

```
char *papiServiceGetServiceName(papi_service_t handle);
```

5.8.3 Inputs

5.8.3.1 handle

Handle to the print service.

5.8.4 Outputs

None

5.8.5 Returns

A pointer to the service name associated with the print service handle. The value returned will be deallocated upon destruction of the service handle.

5.8.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
char *service_name = NULL;
...
service_name = papiServiceGetServiceName(handle);
...
papiServiceDestroy(handle);
```

5.8.7 See Also

papiServiceCreate

5.9 papiServiceGetUserName

5.9.1 Description

Get the user name associated with the print service handle.

5.9.2 Syntax

```
char *papiServiceGetUserName(papi_service_t handle);
```

5.9.3 Inputs

5.9.3.1 handle

Handle to the print service.

5.9.4 Outputs

None

5.9.5 Returns

A pointer to the user name associated with the print service handle.

5.9.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
char *service_name = NULL;
...
user_name = papiServiceGetUserName(handle);
...
papiServiceDestroy(handle);
```

5.9.7 See Also

papiServiceCreate, papiServiceSetUserName

5.10 papiServiceGetPassword

5.10.1 Description

Get the password associated with the print service handle.

5.10.2 Syntax

```
char *papiServiceGetPassword(papi_service_t handle);
```

5.10.3 Inputs

5.10.3.1 handle

Handle to the print service.

5.10.4 Outputs

None

5.10.5 Returns

A pointer to the password associated with the print service handle.

5.10.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
char *password = NULL;
...
password = papiServiceGetPassword(handle);
...
papiServiceDestroy(handle);
```

5.10.7 See Also

papiServiceCreate, papiServiceSetPassword

5.11 papiServiceGetEncryption

5.11.1 Description

Get the encryption associated with the print service handle.

5.11.2 Syntax

```
papi_encryption_t papiServiceGetEncryption(papi_service_t handle);
```

5.11.3 Inputs

5.11.3.1 handle

Handle to the print service.

5.11.4 Outputs

None

5.11.5 Returns

The type of encryption associated with the print service handle.

5.11.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_encryption_t encryption;
...
encryption = papiServiceGetEncryption(handle);
...
papiServiceDestroy(handle);
```

5.11.7 See Also

papiServiceCreate, papiServiceSetEncryption

5.12 papiServiceGetAppData

5.12.1 Description

Get a pointer to the application-specific data associated with the print service handle.

5.12.2 Syntax

```
void *papiServiceGetAppData(papi_service_t handle);
```

5.12.3 Inputs

5.12.3.1 handle

Handle to the print service.

5.12.4 Outputs

None

5.12.5 Returns

A pointer to the application-specific data associated with the print service handle.

5.12.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
void app_data = NULL;
...
app_data = papiServiceGetAppData(handle);
...
papiServiceDestroy(handle);
```

5.12.7 See Also

papiServiceCreate, papiServiceSetAppData

5.13 papiServiceGetAttributeList

5.13.1 Description

Retrieve an attribute list from the print service. This attribute list contains service specific attributes describing service and implementation specific features.

5.13.2 Syntax

```
papi_attribute_t **papiServiceGetAttributeList(papi_service_t handle);
```

5.13.3 Inputs

5.13.3.1 handle

Handle to the print service.

5.13.4 Outputs

None

5.13.5 Returns

An attribute list associated with the print service handle. The attribute list is destroyed when the service handle is destroyed.

5.13.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_attribute_t **attributes = NULL;
...
attributes = papiServiceGetAttributeList(handle);
...
```

```
papiServiceDestroy(handle);
```

5.13.7 See Also

papiServiceCreate, papiServiceDestroy

5.14 papiServiceGetStatusMessage

5.14.1 Description

Get the message associated with the status of the last operation performed. The status message returned from this function may be more detailed than the status message returned from papiStatusString (if the print service supports returning more detailed error messages). The returned message will be localized in the language of the submitter of the original operation.

5.14.2 Syntax

```
Char *papiServiceGetStatusMessage(papi_service_t handle);
```

5.14.3 Inputs

5.14.3.1 handle

Handle to the print service.

5.14.4 Outputs

None

5.14.5 Returns

Pointer to the message associated with the print service handle.

5.14.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
char *message = NULL;
...
message = papiServiceGetStatusMessage(handle);
...
papiServiceDestroy(handle);
```

5.14.7 See Also

papiServiceCreate, papiServiceSetUserName, papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB, papiServiceSetAppData, Printer API, Attributes API, Job API

Chapter 6: Printer API

The printer segment of the PAPI provides a means of interacting with printer objects contained in a print service. This interaction can include listing, querying, modifying, pausing, and releasing the printer objects themselves. It can also include clearing all jobs from a printer object or enumerating all jobs associated with a printer object.

The papiPrinterQuery function queries all/some of the attributes of a printer object. It returns a list of printer attributes. A successful call to papiPrinterQuery is typically followed by code which examines and processes the returned attributes. When the calling program is finished with the printer object and it's attributes, it should then call papiPrinterFree to delete the returned results.

Printers can be found via calls to papiPrintersList. A successful call to papiPrintersList is typically followed by code to iterate through the list of returned printers, possibly querying each (papiPrinterQuery) for further information (e.g. to restrict what printers get displayed for a particular user/request). When the calling program is finished with the list of printer objects, it should then call papiPrinterListFree to free the returned results.

6.1 papiPrintersList

6.1.1 Description

List all printers known by the print service which match the specified filter.
Depending on the functionality of the target service's "printer directory", the returned list may be limited to only printers managed by a particular server or it may include printers managed by other servers.

6.1.2 Syntax

```
papi_status_t papiPrintersList(papi_service_t handle, char *requested_attrs[],
                               papi_filter_t *filter, papi_printer_t **printers );
```

6.1.3 Inputs

6.1.3.1 handle

Handle to the print service.

6.1.3.2 requested_attrs

(optional) NULL terminated 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.)

6.1.3.3 filter

(optional) Pointer to a filter to limit the number if printers returned on the list request. See for details. If NULL is passed then all known printers are listed.

6.1.4 Outputs

6.1.4.1 printers

List of printer objects that matched the filter criteria. The resulting list of printer objects must be deallocated by the caller using papiPrinterListFree().

6.1.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.1.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
char *req_attrs[] = { "printer-name", "printer-uri", NULL };
papi_filter_t filter;
papi_printer_t *printers = NULL;
...
/* Select local printers (non-remote) that support color */
filter.type = PAPI_FILTER_BITMASK;
filter.filter.bitmask.mask  = PAPI_PRINTER_REMOTE |
PAPI_PRINTER_COLOR;
filter.filter.bitmask.value = PAPI_PRINTER_COLOR;
...
status = papiPrinterList(handle, req_attrs, filter, &printers);
...
if (printers != NULL) {
      int i;

      for (i = 0; printers[i] != NULL; i++) {
            ...
      }
      papiPrinterListFree(printers);
}
...
papiServiceDestroy(handle);
```

6.1.7 See Also

papiPrinterListFree, papiPrinterQuery

6.2 papiPrinterQuery

6.2.1 Description

Queries some or all the attributes of the specified printer object. This includes attributes representing information and capabilities of the printer. The caller may use this information 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 implementation specific and is beyond the scope of this standard. The call optionally includes "context" information which specifies job attributes that provide a context that can be used by the print service to construct capabilities information.

6.2.2 Semantics Reference

Get-Printer-Attributes in [RFC2911], section 3.2.5

6.2.3 Syntax

```
papi_status_t papiPrinterQuery(papi_service_t handle, char *name,
                              char  *requested_attrs[], papi_attribute_t **job_attrs,
                              papi_printer_t *printer );
```

6.2.4 Inputs

6.2.4.1 handle

Handle to the print service to use.

6.2.4.2 name

The name or URI of the printer to query.

6.2.4.3 requested_attrs

(optional) NULL terminated 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.)

6.2.4.4 job_attrs

(optional) NULL terminated array of job attributes in the context of which the capabilities information is to be constructed. In other words, the returned printer attributes represent the capabilities of the printer given that these specified job attributes are requested. This allows for more accurate information to be retrieved by the caller for a specific job (e.g. "if the job is printed on A4 size media then duplex output is not available"). If NULL is passed then the full capabilities of the printer are queried.

Support for this argument is optional. If the underlying print system does not have access to capabilities information bound by job context, then this argument may be ignored. But if the calling application will be using the returned information to build print job data, then it is always advisable to specify the job context attributes. The more context information provided, the more accurate capabilities information is likely to be returned from the print system.

6.2.5 Outputs

6.2.5.1 printer

Pointer to a printer object containing the requested attributes.

6.2.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.2.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
char *req_attrs[] = { "printer-name", "printer-uri",
            “printer-state”, “printer-state-reasons”, NULL };
papi_attribute_t **job_attrs = NULL;
papi_printer_t printer = NULL;
...
papiAttributeListAddString(&job_attrs, PAPI_EXCL,
                        "media", "legal");
...
status = papiPrinterQuery(handle, “ipp://server/printers/queue”,
                        req_attrs, job_attrs, &printer);
papiAttributeListFree(job_attrs);
...
if (printer != NULL) {
      /* process the printer object */
      ...
      papiPrinterFree(printer);
}
...
papiServiceDestroy(handle);
```

6.2.8 See Also

papiPrintersList, papiPrinterFree

6.3 papiPrinterModify

6.3.1 Description

Modifies some or all the attributes of the specified printer object. Upon successful completion, the function will return a handle to an object representing the updated printer.

6.3.2 Semantics Reference

Set-Job-Attributes in [RFC3380], section 4.2

6.3.3 Syntax

```
papi_status_t papiPrinterModify(papi_service_t     handle, char *printer_name,
                                papi_attribute_t **attrs, papi_printer_t *printer );
```

6.3.4 Inputs

6.3.4.1 handle

Handle to the print service to use.

6.3.4.2 name

The name or URI of the printer to be modified.

6.3.4.3 attrs

Attributes to be modified. Any attributes not specified are left unchanged. Attributes can be deleted from the print service's printer object through the use of the PAPI_DELETE attribute metadata type.

6.3.5 Outputs

6.3.5.1 printer

The modified printer object.

6.3.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.3.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_printer_t printer = NULL;
```

```
papi_attribute_t **attrs = NULL;
...
papiAttributeListAddString(&attrs, PAPI_EXCL,
                  "printer-location", "Bldg 17/Room 234");
papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
                  “sample-data”, PAPI_DELETE);
...
status = papiPrinterModify(handle, “printer”, attrs, &printer);
...
if (printer != NULL) {
      /* process the printer */
      ...
      papiPrinterFree(printer);
}
...
papiServiceDestroy(handle);
```

6.3.8 See Also

PapiPrinterQuery, papiPrinterAdd, papiPrinterRemove, papiPrinterFree

6.4 papiPrinterAdd

6.4.1 Description

Creates a printer object with some or all the attributes specified. Upon successful completion, the function will return a handle to an object representing the created printer.

6.4.2 Semantics Reference

Set-Job-Attributes in [RFC3380], section 4.2

6.4.3 Syntax

```
papi_status_t papiPrinterAdd(papi_service_t     handle, char *printer_name,
                            papi_attribute_t **attrs, papi_printer_t *printer );
```

6.4.4 Inputs

6.4.4.1 handle

Handle to the print service on which to create the printer object.

6.4.4.2 name

The name or URI of the printer to be created

6.4.4.3 attrs

Attributes to associate with the printer object being created. The print service may not honor all requested attributes and it may also add attributes of it's own during printer creation.

6.4.5 Outputs

6.4.5.1 printer

The created printer object.

6.4.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.4.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_printer_t printer = NULL;
papi_attribute_t **attrs = NULL;
...
papiAttributeListAddString(&attrs, PAPI_ATTR_EXCL,
                  "printer-location", "Bldg 17/Room 234");
papiAttributeListAddString(&attrs, PAPI_ATTR_APPEND,
                  “document-format-supported”, “application/ps”);
papiAttributeListAddString(&attrs, PAPI_ATTR_APPEND,
                  “document-format-supported”, “text/plain”);
papiAttributeListAddInteger(&attrs, PAPI_ATTR_APPEND,
                  “copies-default”, 3);
...
status = papiPrinterModify(handle, “ipp://server/printers/triplicate”,
                  attrs, &printer);
papiAttributeListFree(attrs);
...
if (status != PAPI_OK) {
      /* report a failure */
}

if (printer != NULL) {
      /* process the printer */
      ...
      papiPrinterFree(printer);
}
...
papiServiceDestroy(handle);
```

6.4.8 See Also

papiPrinterQuery, papiPrinterModify, papiPrinterRemove, papiPrinterFree

6.5 papiPrinterRemove

6.5.1 Description

Removes a printer object from a print service.

6.5.2 Semantics Reference

Set-Job-Attributes in [RFC3380], section 4.2

6.5.3 Syntax

```
papi_status_t papiPrinterRemove(papi_service_t handle, char *printer_name);
```

6.5.4 Inputs

6.5.4.1 handle

Handle to the print service from which to remove the printer object.

6.5.4.2 name

The name or URI of the printer to be removed

6.5.5 Outputs

None

6.5.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.5.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiPrinterRemove(handle, “ipp://server/printers/goodbye”);
if (status != PAPI_OK) {
      /* report a failure */
}
...
papiServiceDestroy(handle);
```

6.5.8 See Also

papiPrinterQuery, papiPrinterModify, papiPrinterAdd, papiPrinterFree

6.6 papiPrinterPause

6.6.1 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 papiPrinterResume to undo the effects of this operation.

6.6.2 Semantics Reference

Pause-Printer in [RFC2911], section 3.2.7

6.6.3 Syntax

```
papi_status_t papiPrinterPause(papi_service_t handle, char *name,
                                   char *message );
```

6.6.4 Inputs

6.6.4.1 handle

Handle to the print service to use.

6.6.4.2 name

The name or URI of the printer to operate on.

6.6.4.3 message

(optional) An explanatory message to be associated with the paused printer. This message may be ignored if the underlying print system does not support associating a message with a paused printer.

6.6.5 Outputs

None

6.6.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.6.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiPrinterPause(handle, “printer”, “because I can”);
...
papiServiceDestroy(handle);
```

6.6.8 See Also

papiPrinterResume

6.7 papiPrinterResume

6.7.1 Description

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

6.7.2 Semantics Reference

Resume-Printer in [RFC2911], section 3.2.8

6.7.3 Syntax

```
papi_status_t papiPrinterResume(papi_service_t handle, char *name);
```

6.7.4 Inputs

6.7.4.1 handle

Handle to the print service to use.

6.7.4.2 name

The name or URI of the printer to operate on.

6.7.5 Outputs

None

6.7.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.7.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiPrinterResume(handle, “printer”);
...
papiServiceDestroy(handle);
```

6.7.8 See Also

papiPrinterPause

6.8 papiPrinterEnable

6.8.1 Description

Requests that the printer enable job creation (queueing) (i.e. it undoes the effects of papiPrinterDisable). This operation is optional and may not be supported by all printers/servers, but it must be supported if papiPrinterDisable is supported.

6.8.2 Semantics Reference

Resume-Printer in [RFC2911], section 3.2.8

6.8.3 Syntax

```
papi_status_t papiPrinterEnable(papi_service_t handle, char *name);
```

6.8.4 Inputs

6.8.4.1 handle

Handle to the print service to use.

6.8.4.2 name

The name or URI of the printer to operate on.

6.8.5 Outputs

None

6.8.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.8.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiPrinterEnable(handle, “printer”);
...
papiServiceDestroy(handle);
```

6.8.8 See Also

papiPrinterDisable

6.9 papiPrinterDisable

6.9.1 Description

Requests that the printer disable job creation (queueing) (i.e. it undoes the effects of papiPrinterEnable). This operation is optional and may not be supported by all printers/servers, but it must be supported if papiPrinterEnable is supported.

6.9.2 Semantics Reference

Resume-Printer in [RFC2911], section 3.2.8

6.9.3 Syntax

```
papi_status_t papiPrinterEnable(papi_service_t handle, char *name, char *message);
```

6.9.4 Inputs

6.9.4.1 handle

Handle to the print service to use.

6.9.4.2 name

The name or URI of the printer to operate on.

6.9.4.3 Message

An optional reason for disabling the print queue.

6.9.5 Outputs

None

6.9.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.9.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiPrinterEnable(handle, “printer”, “because it's tuesday”);
...
papiServiceDestroy(handle);
```

6.9.8 See Also

papiPrinterDisable

6.10 papiPrinterPurgeJobs

6.10.1 Description

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

6.10.2 Semantics Reference

Purge-Jobs in [RFC2911], section 3.2.9

6.10.3 Syntax

```
papi_status_t papiPrinterPurgeJobs(papi_service_t handle, char *name,
                                   papi_job_t **jobs);
```

6.10.4 Inputs

6.10.4.1 handle

Handle to the print service to use.

6.10.4.2 name

The name or URI of the printer to operate on.

6.10.5 Outputs

6.10.5.1 jobs

(optional) Pointer to a list of purged jobs with the identifying information (job-id/job-uri), success/fail, and possibly a detailed message. If NULL is passed then no job list is returned. Support for the returned job list is optional and may not be supported by all implementations (if not supported, the function completes with PAPI_OK_SUBST but no list is returned).

6.10.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.10.7 Example

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
char *service_name = "ipp://printserv:631";
char *user_name = "pappy";
char *password = "goober";
char *printer_name = "my-printer";
papi_job_t *jobs = NULL;

status = papiServiceCreate(handle, service_name, user_name,
                        password, NULL, PAPI_ENCRYPT_IF_REQUESTED,
                        NULL);
if (status != PAPI_OK) {
      /* handle the error */
      ...
}

status = papiPrinterPurgeJobs(handle, printer_name, &jobs);
if (status != PAPI_OK) {
      /* handle the error */
      fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
                  papiServiceGetStatusMessage(handle));
      ...
}

if (jobs != NULL) {
      int i;

      for(i=0; jobs[i] != NULL; i++) {
            /* process the job */
            ...
      }
```

```
    papiJobListFree(jobs);
}
...

papiServiceDestroy(handle);
```

6.10.8 See Also

papiJobCancel, papiJobListFree

6.11 papiPrinterListJobs

6.11.1 Description

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

6.11.2 Semantics Reference

Get-Jobs in [RFC2911], section 3.2.6

6.11.3 Syntax

```
papi_status_t papiPrinterListJobs(papi_service_t handle, char *printer,
                                  char *requested_attrs[], int type_mask,
                                  int max_num_jobs, papi_job_t **jobs );
```

6.11.4 Inputs

6.11.4.1 handle

Handle to the print service to use.

6.11.4.2 name

The name or URI of the printer to query.

6.11.4.3 requested_attrs

(optional) NULL terminated array of attributes to be queried. If NULL is passed then all available 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.)

6.11.4.4 type_mask

A bit mask which determines what jobs will get returned. The following constants can be bitwise-OR-ed together to select which types of jobs to list:

```
#define PAPI_LIST_JOBS_OTHERS        0x0001 /* return jobs other than
                                               those submitted by the
                                               user name associated with
                                               the handle */
#define PAPI_LIST_JOBS_COMPLETED     0x0002 /* return completed jobs */
#define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
                                               jobs */
#define PAPI_LIST_JOBS_ALL           0xFFFF /* return all jobs */
```

6.11.4.5 max_num_jobs

Limit to the number of jobs returned. If 0 is passed, then there is no limit to the number of jobs which may be returned.

6.11.5 Outputs

6.11.5.1 jobs

List of job objects returned.

6.11.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

6.11.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_job_t *jobs = NULL;
char *job_attrs[] = {
      “job-id”, “job-name”, “job-originating-user-name”,
      “job-state”, “job-state-reasons”, “job-state-message” };
...
status = papiPrinterListJobs(handle, printer_name, job_attrs,
                        PAPI_LIST_JOBS_ALL, 0, &jobs);
...
if (jobs != NULL) {
      int i;

      for(i = 0; jobs[i] != NULL; i++) {
            /* process the job */
            ...
      }
      papiJobListFree(jobs);
}
...
papiServiceDestroy(handle);
```

6.11.8 See Also

papiJobQuery, papiJobListFree

6.12 papiPrinterGetAttributeList

6.12.1 Description

Get the attribute list associated with a printer object.
This function retrieves an attribute list from a printer object returned in a previous call.
Printer objects are returned as the result of operations performed by papiPrintersList, papiPrinterQuery, and papiPrinterModify.

6.12.2 Syntax

```
papi_attribute_t **papiPrinterGetAttributeList(papi_printer_t printer );
```

6.12.3 Inputs

6.12.3.1 printer

Handle of the printer object.

6.12.4 Outputs

none

6.12.5 Returns

Pointer to the attribute list associated with the printer object. This attribute list is deallocated when the printer object it was retrieved from is deallocated using papiPrinterFree(printer).

6.12.6 Example

```
papi_attribute_t **attrs = NULL;
papi_printer_t printer = NULL;
...
attrs = papiPrinterGetAttributeList(printer);
...
papiPrinterFree(printer);
```

6.12.7 See Also

papiPrintersList, papiPrinterQuery, papiPrinterModify

6.13 papiPrinterFree

6.13.1 Description

Free a printer object.

6.13.2 Syntax

```
void papiPrinterFree(papi_printer_t printer);
```

6.13.3 Inputs

6.13.3.1 printer

Handle of the printer object to free.

6.13.4 Outputs

none

6.13.5 Returns

none

6.13.6 Example

```
papi_printer_t printer = NULL;
...
papiPrinterFree(printer);
```

6.13.7 See Also

papiPrinterQuery, papiPrinterModify

6.14 papiPrinterListFree

6.14.1 Description

Free a list of printer objects.

6.14.2 Syntax

```
void papiPrinterListFree(papi_printer_t *printers);
```

6.14.3 Inputs

6.14.3.1 printers

Pointer to the printer object list to free.

6.14.4 Outputs

none

6.14.5 Returns

none

6.14.6 Example

```
papi_printer_t* printers = NULL;
...
papiPrinterListFree(printers);
```

6.14.7 See Also

papiPrintersList

Chapter 7: Job API

The job segment of the PAPI provides a means of interacting with job objects contained in a print service. This interaction can include listing, querying, creating, modifying, canceling, holding, releasing, and restarting the job objects themselves.

The papiJobSubmit, papiJobSubmitByReference, papiJobStreamOpen, and papiJobStreamClose functions provide a means of creating job objects under a print service. The papiJobValidate function can be used to determine if a job submission will be successful. Each of these functions results in a job object with an attribute list that can be queried to determine what the resulting job looks like.

The papiJobQuery function queries all/some of the attributes of a job. A successful call to papiJobQuery is typically followed by code which examines and processes the returned attributes. When the calling program is finished with the job object and it's attributes, it should then call papiJobFree to delete the returned results.

Jobs and job state can be modified through the use of papiJobModify, papiJobHold, papiJobRelease, and papiJobRestart. The papiJobModify call returns a job object that contains a representation of the modified job. The job object's attribute list can be queried to determin what the resulting job looks like. When the calling program is finished with the job object and it's attributes, it should then call papiJobFree to delete the returned results.

7.1 papiJobSubmit

7.1.1 Description

Submits a print job having the specified attributes to the specified printer. This interface copies the specified print files before returning to the caller (contrast to papiJobSubmitByReference). The caller must call papiJobFree when done in order to free the resources associated with the returned job object. Attributes of the print job may be passed in the job_attributes argument and/or in a job ticket (using the job_ticket argument). If both are specified, the attributes in the job_attributes list will be applied to the job_ticket attributes and the resulting attribute set will be used.

7.1.2 Semantics Reference

Print-Job in [RFC2911], section 3.2.1

7.1.3 Syntax

```
papi_status_t papiJobSubmit(papi_service_t handle, char *printer_name,
                            papi_attribute_t **job_attributes,
                            papi_job_ticket_t *job_ticket,
                            char **file_names, papi_job_t *job );
```

7.1.4 Inputs

7.1.4.1 handle

Handle to the print service to use.

7.1.4.2 printer_name

Pointer to the name of the printer to which the job is to be submitted.

7.1.4.3 job_attributes

(optional) The list of attributes describing the job and how it is to be printed. 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 submitted with the job.

7.1.4.4 job_ticket

(optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no job ticket is used with the job. Whether the implementation passes both the attributes and the job ticket to the server/printer, or merges them to some print protocol or internal representation depends on the implementation.

7.1.4.5 file_names

NULL terminated list of pointers to names of files to print. If more than one file is specified, the files will be treated by the print system as separate "documents" for things like page breaks and separator sheets, but they will be scheduled and printed together as one job and the specified attributes will apply to all the files.
These file names may contain absolute path names or relative path names (relative to the current path). The implementation MUST copy the file contents before returning.

7.1.5 Outputs

7.1.5.1 job

The resulting job object representing the submitted job. The caller must deallocate this object using papiJobFree() when finished using it.

7.1.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.1.7 Example

```
papi_status_t status;
```

```
papi_service_t handle = NULL;
papi_attribute_t **attrs = NULL;
papi_job_ticket_t *ticket = NULL;
char *files[] = { "/etc/motd", NULL };
papi_job_t job = NULL;
...
papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
                        PAPI_STRING, 1, "test job");
papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
                        PAPI_INTEGER, 4);
...
status = papiJobSubmit(handle, “printer”, attrs, ticket, files, &job);
papiAttributeListFree(attrs);
...
if (job != NULL) {
      /* look at the job object (maybe get the id) */
      papiJobFree(job);
}
...
```

7.1.8 See Also

papiJobSubmitByReference, papiJobValidate, papiJobStreamOpen, papiJobStreamWrite, papiJobStreamClose, papiJobFree

7.2 papiJobSubmitByReference

7.2.1 Description

Submits a print job having the specified attributes to the specified printer. This interface delays copying the specified print files as long as possible, ideally only "pulling" the files when the printer is actually printing the job (contrast to papiJobSubmit).
Attributes of the print job may be passed in the job_attributes argument and/or in a job ticket (using the job_ticket argument). If both are specified, the attributes in the job_attributes list will be applied to the job_ticket attributes and the resulting attribute set will be used.

7.2.2 Semantics Reference

Print-URI in [RFC2911], section 3.2.2

7.2.3 Syntax

```
papi_status_t papiJobSubmitByReference(papi_service_t handle,
                        char *printer_name,
                        papi_attribute_t **job_attributes,
                        papi_job_ticket_t *job_ticket,
char **file_names, papi_job_t *job );
```

7.2.4 Inputs

7.2.4.1 handle

Handle to the print service to use.

7.2.4.2 printer_name

Pointer to the name of the printer to which the job is to be submitted.

7.2.4.3 job_attributes

(optional) The list of attributes describing the job and how it is to be printed. 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 submitted with the job.

7.2.4.4 job_ticket

(optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no job ticket is used with the job. Whether the implementation passes both the attributes and the job ticket to the server/printer, or merges them to some print protocol or internal representation depends on the implementation.

7.2.4.5 file_names

NULL terminated list of pointers to names of files to print. If more than one file is specified, the files will be treated by the print system as separate "documents" for things like page breaks and separator sheets, but they will be scheduled and printed together as one job and the specified attributes will apply to all the files.
These file names may contain absolute path names, relative path names or URIs ([RFC1738], [RFC2396]). The implementation SHOULD NOT copy the referenced data unless (or until) it is no longer feasible to maintain the reference. Feasibility limitations may arise out of security issues, name space issues, and/or protocol or printer limitations. Implementations MUST support the absolute path, relative path, and "file:" URI scheme. Use of other URI schemes could result in a PAPI_URI_SCHEME error, depending on the implementation.
The semantics explained in the preceding paragraphs allows for flexibility in the PAPI implementation. For example: (1) PAPI on top of a local service to maintain the reference for the life of the job, if the local service supports it. (2) PAPI on top of IPP to send a reference when the server can access the referenced data and copy it when it is not accessible to the server. (3) PAPI on top of network printing protocols that don't support references to copy the data on the way out to the remote server.

7.2.5 Outputs

7.2.5.1 job

The resulting job object representing the submitted job. The caller must deallocate this object using papiJobFree() when finished using it.

7.2.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.2.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_attribute_t **attrs = NULL;
papi_job_ticket_t *ticket = NULL;
char *files[] = { "/etc/motd", NULL };
papi_job_t job = NULL;
...
papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
                        PAPI_STRING, 1, "test job");
papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
                        PAPI_INTEGER, 4);

status = papiJobSubmitByReference(handle, “printer”, attrs, ticket,
                   files, &job);
papiAttributeListFree(attrs)
...
if (job != NULL) {
      /* look at the job object (maybe get the id) */
      papiJobFree(job);
}
...
```

7.2.8 See Also

papiJobSubmit, papiJobValidate, papiJobStreamOpen, papiJobStreamWrite, papiJobStreamClose, papiJobFree

7.3 papiJobValidate

7.3.1 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. Attributes of the print job may be passed in the job_attributes argument and/or in a job ticket (using the job_ticket argument). If both are specified, the attributes in the job_attributes list will be applied to the job_ticket attributes and the resulting attribute set will be used.

7.3.2 Semantics Reference

Validate-Job in [RFC2911], section 3.2.3

7.3.3 Syntax

```
papi_status_t papiJobValidate(papi_service_t handle, char *printer_name,
                              papi_attribute_t **job_attributes,
                              papi_job_ticket_t *job_ticket,
                              char **file_names, papi_job_t *job );
```

7.3.4 Inputs

7.3.4.1 handle

Handle to the print service to use.

7.3.4.2 printer_name

Pointer to the name of the printer to which the job is to be validated.

7.3.4.3 job_attributes

(optional) The list of attributes describing the job and how it is to be printed. 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 submitted with the job.

7.3.4.4 job_ticket

(optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no job ticket is used with the job. Whether the implementation passes both the attributes and the job ticket to the server/printer, or merges them to some print protocol or internal representation depends on the implementation.

7.3.4.5 file_names

NULL terminated list of pointers to names of files to validate.

7.3.5 Outputs

7.3.5.1 job

The resulting job object representing the validated job. The caller must deallocate this object using papiJobFree() when finished using it.

7.3.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.3.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_attribute_t **attrs = NULL;
papi_job_ticket_t *ticket = NULL;
char *files[] = { "/etc/motd", NULL };
papi_job_t job = NULL;
...
papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
                        PAPI_STRING, 1, "test job");
papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
                        PAPI_INTEGER, 4);
...
status = papiJobValidate(handle, printer, attrs, ticket, files, &job);
papiAttributeListFree(attrs);
...
if (job != NULL) {
     papiJobFree(job);
}
...
```

7.3.8 See Also

papiJobSubmit, papiJobSubmitByReference, papiJobStreamOpen, papiJobStreamWrite, papiJobStreamClose, papiJobFree

7.4 papiJobStreamOpen

7.4.1 Description

Opens a print job and an associated stream of print data to be sent to the specified printer. After calling this function papiJobStreamWrite can be called (repeatedly) to write the print data to the stream, and then papiJobStreamClose is called to complete the submission of the print job.
After this function is called successfully, papiJobStreamClose must eventually be called to close the stream (this includes all error paths).
Attributes of the print job may be passed in the job_attributes argument and/or in a job ticket (using the job_ticket argument). If both are specified, the attributes in the job_attributes list will be applied to the job_ticket attributes and the resulting attribute set

will be used.

7.4.2 Syntax

```
papi_status_t papiJobStreamOpen(papi_service_t handle, char *printer_name,
                                papi_attribute_t **job_attributes,
                                papi_job_ticket_t *job_ticket,
                                papi_stream_t *stream);
```

7.4.3 Inputs

7.4.3.1 handle

Handle to the print service to use.

7.4.3.2 printer_name

Pointer to the name of the printer to which the job is to be validated.

7.4.3.3 job_attributes

(optional) The list of attributes describing the job and how it is to be printed. 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 submitted with the job.

7.4.3.4 job_ticket

(optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no job ticket is used with the job. Whether the implementation passes both the attributes and the job ticket to the server/printer, or merges them to some print protocol or internal representation depends on the implementation.

7.4.4 Outputs

7.4.4.1 stream

The resulting stream object to which print data can be written. The stream object will be deallocated when closed using papiJobStreamClose().

7.4.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.4.6 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_attribute_t **attrs = NULL;
papi_job_ticket_t *ticket = NULL;
papi_job_t job = NULL;
char buffer[4096];
size_t buflen = 0;
...
papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
                        PAPI_STRING, 1, "test job");
papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
                        PAPI_INTEGER, 4);
...
status = papiJobStreamOpen(handle, “printer”, attrs, ticket, &stream);
papiAttributeListFree(attrs);
...
while (print_data_remaining) {
      status = papiJobStreamWrite(handle, stream, buffer, buflen);
}
...
status = papiJobStreamClose(handle, stream, &job);
...
if (job != NULL) {
      ...
      papiJobFree(job);
}
...
```

7.4.7 See Also

papiJobStreamWrite, papiJobStreamClose

7.5 papiJobStreamWrite

7.5.1 Description

Writes print data to the specified open job stream. The open job stream must have been obtained by a successful call to papiJobStreamOpen

7.5.2 Syntax

```
papi_status_t papiJobStreamWrite(papi_service_t handle, papi_stream_t stream,
                        void *buffer, size_t buflen);
```

7.5.3 Inputs

7.5.3.1 handle

Handle to the print service to use.

7.5.3.2 stream

The open stream object to which print data is written.

7.5.3.3 buffer

Pointer to the buffer of print data to write.

7.5.3.4 buflen

The number of bytes to write.

7.5.4 Outputs

none

7.5.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.5.6 Example

```
See papiJobStreamOpen
```

7.5.7 See Also

papiJobStreamOpen, papiJobStreamClose

7.6 papiJobStreamClose

7.6.1 Description

Closes the specified open job stream and completes submission of the job (if there were no previous errors returned from papiJobSubmitWrite). The open job stream must have been obtained by a successful call to papiJobStreamOpen.

7.6.2 Syntax

```
papi_status_t papiJobStreamClose(papi_service_t handle, papi_stream_t stream,
                                 papi_job_t *job);
```

7.6.3 Inputs

7.6.3.1 handle

Handle to the print service to use.

7.6.3.2 stream

The open stream object to close.

7.6.4 Outputs

7.6.4.1 Job

The resulting job object representing the submitted job. The caller must deallocate this object using papiJobFree() when finished using it.

7.6.5 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.6.6 Example

```
See papiJobStreamOpen
```

7.6.7 See Also

papiJobStreamOpen, papiJobStreamWrite

7.7 papiJobQuery

7.7.1 Description

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

7.7.2 Semantics Reference

Get-Job-Attributes in [RFC2911], section 3.3.4

7.7.3 Syntax

```
papi_status_t papiJobQuery(papi_service_t handle,char* printer_name,
                           int32_t job_id, char *requested_attrs[],
                           papi_job_t *job);
```

7.7.4 Inputs

7.7.4.1 handle

Handle to the print service to use.

7.7.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.7.4.3 job_id

The ID number of the job to be queried.

7.7.4.4 requested_attrs

NULL terminated array of attributes to be queried. If NULL is passed then all available 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.)

7.7.5 Outputs

7.7.5.1 job

The returned job object containing the requested attributes.

7.7.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.7.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_job_t job = NULL;
char *job_attrs[] = {
      "job-id", "job-name", "job-originating-user-name",
      "job-state", "job-state-reasons", NULL };
...
status = papiJobQuery(handle, “printer”, job_id, job_attrs, &job);
...
if (job != NULL) {
      /* process the job */
      ...
      papiJobFree(job);
}
...
```

7.7.8 See Also

papiPrinterListJobs, papiJobFree

7.8 papiJobModify

7.8.1 Description

Modifies some or all the attributes of the specified job object. Upon successful completion, the function will return a handle to an object representing the updated job.

7.8.2 Semantics Reference

Set-Job-Attributes in [RFC3380], section 4.2

7.8.3 Syntax

```
papi_status_t papiJobModify(papi_service_t handle,char* printer_name,
                            int32_t job_id, papi_attribute_t **attrs,
                            papi_job_t *job);
```

7.8.4 Inputs

7.8.4.1 handle

Handle to the print service to use.

7.8.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.8.4.3 job_id

The ID number of the job to be queried.

7.8.4.4 attrs

Attributes to be modified. Any attributes not specified are left unchanged.

7.8.5 Outputs

7.8.5.1 job

The modified job object.

7.8.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.8.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
papi_job_t job = NULL;
papi_attribute_t **attrs = NULL;
...
papiAttributeListAddString(&attrs, PAPI_EXCL,
                  "job-name", "sample job");
papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
                  “media”, PAPI_DELETE);
...
status = papiJobModify(handle, “printer”, 12, attrs, &job);
...
if (job != NULL) {
      /* process the job */
      ...
      papiJobFree(job);
}
...
```

7.8.8 See Also

papiJobFree

7.9 papiJobCancel

7.9.1 Description

Cancel the specified print job

7.9.2 Semantics Reference

Cancel Job in [RFC2911], section 3.3.3

7.9.3 Syntax

```
papi_status_t papiJobCancel(papi_service_t handle, char* printer_name,
                  int32_t job_id);
```

7.9.4 Inputs

7.9.4.1 handle

Handle to the print service to use.

7.9.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.9.4.3 job_id

The ID number of the job to be canceled.

7.9.5 Outputs

none

7.9.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.9.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiJobCancel(handle, “printer”, 12);
...
```

7.9.8 See Also

papiPrinterPurgeJobs

7.10 papiJobHold

7.10.1 Description

Hold the specified print job

7.10.2 Semantics Reference

Hold Job in [RFC2911], section 3.3.5

7.10.3 Syntax

```
papi_status_t papiJobHold(papi_service_t handle, char* printer_name,
                          int32_t job_id);
```

7.10.4 Inputs

7.10.4.1 handle

Handle to the print service to use.

7.10.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.10.4.3 job_id

The ID number of the job to be held.

7.10.5 Outputs

none

7.10.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.10.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiJobHold(handle, “printer”, 12);
...
```

7.10.8 See Also

papiJobRelease

7.11 papiJobRelease

7.11.1 Description

Release the specified print job

7.11.2 Semantics Reference

Release Job in [RFC2911], section 3.3.6

7.11.3 Syntax

```
papi_status_t papiJobRelease(papi_service_t handle,char* printer_name,
                              int32_t job_id);
```

7.11.4 Inputs

7.11.4.1 handle

Handle to the print service to use.

7.11.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.11.4.3 job_id

The ID number of the job to be released.

7.11.5 Outputs

none

7.11.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.11.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiJobRelease(handle, “printer”, 12);
...
```

7.11.8 See Also

papiJobHold

7.12 papiJobRestart

7.12.1 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.

7.12.2 Semantics Reference

Restart Job in [RFC2911], section 3.3.7

7.12.3 Syntax

```
papi_status_t papiJobRestart(papi_service_t handle,char* printer_name,
                             int32_t job_id);
```

7.12.4 Inputs

7.12.4.1 handle

Handle to the print service to use.

7.12.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.12.4.3 job_id

The ID number of the job to be restart.

7.12.5 Outputs

none

7.12.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.12.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiJobRestart(handle, “printer”, 12);
...
```

7.12.8 See Also

papiJobHold, papiJobRelease

7.13 papiJobPromote

7.13.1 Description

Promotes a job to the front of the queue so that it may be printed after any currently printing job completes. This operation is optional and may not be supported by all printers/servers.

7.13.2 Semantics Reference

Restart Job in [RFC2911], section 3.3.7

7.13.3 Syntax

```
papi_status_t papiJobPromote(papi_service_t handle,char* printer_name,
                             int32_t job_id);
```

7.13.4 Inputs

7.13.4.1 handle

Handle to the print service to use.

7.13.4.2 printer_name

Pointer to the name or URI of the printer to which the job was submitted.

7.13.4.3 job_id

The ID number of the job to be promoted.

7.13.5 Outputs

none

7.13.6 Returns

If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

7.13.7 Example

```
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiJobPromote(handle, “printer”, 12);
...
```

7.13.8 See Also

papiJobHold, papiJobRelease

7.14 papiJobGetAttributeList

7.14.1 Description

Get the attribute list associated with a job object.
This function retrieves an attribute list from a job object returned in a previous call. Job objects are returned as a result of the operations performed by papiPrinterListJobs, papiJobQuery, papiJobModify, papiJobSubmit, papiJobSubmitByReference, papiJobValidate, and papiJobStreamClose.

7.14.2 Syntax.

```
papi_attribute_t **papiJobGetAttributeList(papi_job_t job);
```

7.14.3 Inputs

7.14.3.1 job

Handle of the job object.

7.14.4 Outputs

none

7.14.5 Returns

The attribute list associated with the job object. The attribute list is deallocated when the containing job object is destroyed using papiJobFree().

7.14.6 Example

```
papi_job_t job = NULL;
papi_attribute_list **attrs = NULL;
...
attrs = papiJobGetAttributeList(job);
...
papiJobFree(job);
```

7.14.7 See Also

papiPrinterListJobs, papiJobQuery, papiJobModify, papiJobSubmit, papiJobSubmitByReference, papiJobValidate, papiJobStreamClose

7.15 papiJobGetPrinterName

7.15.1 Description

Get the printer name associated with a job object.

7.15.2 Syntax.

```
char *papiJobGetPrinterName(papi_job_t job);
```

7.15.3 Inputs

7.15.3.1 job

Handle of the job object.

7.15.4 Outputs

none

7.15.5 Returns

Pointer to the printer name associated with the job object. The resulting string is deallocated when the containing job object is destroyed using papiJobFree().

7.15.6 Example

```
char *printer = NULL;
papi_job_t job = NULL;
...
printer = papiJobGetPrinterName(job);
...
papiJobFree(job);
```

7.15.7 See Also

papiJobGetAttributeList

7.16 papiJobGetId

7.16.1 Description

Get the job ID associated with a job object.

7.16.2 Syntax.

```
int32_t papiJobGetId(papi_job_t job);
```

7.16.3 Inputs

7.16.3.1 job

Handle of the job object.

7.16.4 Outputs

none

7.16.5 Returns

The job id associated with the job object.

7.16.6 Example

```
papi_job_t job = NULL;
int32_t id;
...
id = papiJobGetId(job);
...
papiJobFree(job);
```

7.16.7 See Also

papiJobGetAttributeList

7.17 papiJobGetJobTicket

7.17.1 Description

Get the job ticket associated with a job object. The job ticket is deallocated when the containing job object is destroyed using papiJobFree().

7.17.2 Syntax

```
papi_job_ticket_t *papiJobGetJobTicket(papi_job_t job);
```

7.17.3 Inputs

7.17.3.1 job

Handle of the job object.

7.17.4 Outputs

none

7.17.5 Returns

Pointer to the job ticket associated with the job object.

7.17.6 Example

```
papi_job_t job = NULL;
papi_job_ticket_t *ticket;
...
ticket = papiJobGetJobTicket(job);
...
papiJobFree(job);
```

7.17.7 See Also

papiJobSubmit, papiJobSubmitByReference, papiJobValidate, papiJobStreamOpen

7.18 papiJobFree

7.18.1 Description

Free a job object.

7.18.2 Syntax

```
void papiJobFree(papi_job_t job );
```

7.18.3 Inputs

7.18.3.1 Job

Handle of the job object to free.

7.18.4 Outputs

none

7.18.5 Returns

none

7.18.6 Example

```
papi_job_t job = NULL;
```

```
...
papiJobFree(job);
```

7.18.7 See Also

papiJobSubmit, papiJobSubmitByReference, papiJobValidate, papiJobStreamClose, papiJobQuery, papiJobModify

7.19 papiJobListFree

7.19.1 Description

Free a job list.

7.19.2 Syntax

```
void papiJobListFree(papi_job_t *job );
```

7.19.3 Inputs

7.19.3.1 Job

Handle of the job list to free.

7.19.4 Outputs

none

7.19.5 Returns

none

7.19.6 Example

```
papi_job_t *jobs = NULL;
...
papiJobListFree(jobs);
```

7.19.7 See Also

papiPrinterListJobs

Chapter 8: Miscellaneous API

8.1 papiStatusString

8.1.1 Description

Get a status string for the specified papi_status_t. The status message returned from this function may be less detailed than the status message returned from papiServiceGetStatusMessage (if the print service supports returning more detailed error messages)

8.1.2 Syntax

```
char *papiStatusString(papi_status_t status);
```

8.1.3 Inputs

8.1.3.1 status

The status value to convert to a status string

8.1.4 Outputs

none

8.1.5 Returns

The returned string provides a (potentially localized) human readable message representing the status provided. The return value should not be deallocated by the caller.

8.1.6 Example

```
papi_status_t status;
char *message;
...
message = papiServiceGetStatusMessage(handle);
...
```

8.1.7 See Also

PapiServiceGetStatusMessage

8.2 papiLibrarySupportedCalls

8.2.1 Description

The papiLibrarySupportedCalls() function can be called to request a list of API functions that are supported in the implementation. Support for a function means that the implementation of that function is not a stub that simply returns PAPI_OPERATION_NOT_SUPPORTED

8.2.2 Syntax

```
char **papiLibrarySupportedCalls();
```

8.2.3 Inputs

none

8.2.4 Outputs

none

8.2.5 Returns

A NULL terminated list of supported function names. This list should not be deallocated by the caller.

8.2.6 Example

```
papi_service_t handle = NULL;
char **calls;
...
calls = papiLibrarySupportedCalls(handle);
...
```

8.2.7 See Also

Conformance Table

8.3 papiLibrarySupportedCall

8.3.1 Description

The papiLibrarySupportedCalls() function can be called to request a list of API functions that are supported in the implementation. Support for a function means that the implementation of that function is not a stub that simply returns PAPI_OPERATION_NOT_SUPPORTED

8.3.2 Syntax

```
char papiLibrarySupportedCall(char *name);
```

8.3.3 Inputs

8.3.3.1 name

The name of the function that is being asked about

8.3.4 Outputs

none

8.3.5 Returns

A return of PAPI_TRUE indicates that the named function is supported by the API implementation. A return of PAPI_FALSE indicates that the the named function is not supported by the API implementation.

8.3.6 Example

```
papi_service_t handle = NULL;
char supported;
...
supported = papiLibrarySupportedCall(handle, “papiJobQuery”);
...
```

8.3.7 See Also

ConformanceProfiles

Chapter 9: Capabilities

9.1 Introduction

In the context of this document, printer capabilities refers to information about the features, options, limitation, etc. of a print device (either an actual device or an abstract device which may represent a group or pool of actual devices). This includes such information as:

- Does the printer support color printing?
- At what resolution(s) can the printer print?
- What input trays are present?
- What size media is loaded in each tray?
- Which trays are manual-feed and which are auto-feed?
- Can the printer print duplex output?
- What is the printable area on each of the loaded media?
- What output bins are present?
- What finishing (staple, punch, etc.) does the printer support?
- What combinations of features are not allowed together?
- What features should be presented on the print user interface?
- ... and many others...

The uses of printer capabilities by applications include:

To control how to display print options in a print UI dialog. Examples:

- What values to put in the binselection pull-down lists
- Whether or not to gray-out the duplex option when a particular output bin has been selected.
- Whether or not to display a color vs. black and white selection

To Control how the printdata stream is generated. Examples:

- How large an image to draw to fill the printable area.
- How much to shift the image if “3-hole punch” finishing hasbeen selected.
- How to request that the printer print on paper from the manual envelope feeder

To do job validation andprinter selection. Examples:

- Can I print this job with these options on this printer?

- Find a printer which can print this job with these options.

9.2 Objectives

This section attempts to describe the objectives of the PAPI printer capabilities support. It is important to understand these objectives in order to understand why the support is structured the way that it is.

9.2.1 Standard printer capabilities API

There is no standard API which a Linux application can use to retrieve printer capabilities regardless of the device, driver, and print server being used. This makes it very difficult for application writers to support generating print data without writing multiple versions of the print logic or without tying the application to very specific print system environments. This specification provides the standard API, making applications which use it independent of the underlying print system.

9.2.2 Independent of underlying source of capabilities

The capabilities information returned to the application may come from one of a variety of sources or combination of sources. The data retrieved from these sources may be represented in a variety of formats, including:

- PPD files
- UPDF database
- SNMP queries
- Device drivers

The API defined here hides these differences so that the application is independent of data source and format used.

9.2.3 Support returning information in context

The API supports a means for requesting capabilities information in the context of a particular set of job options. For example, set of printer capabilities can be queried given that medium and color/black-and-white selections have already been made.

9.2.4 Support returning constraints

The API must support a means for returning constraints on printer capabilities. This allows applications to not submit job with disallowed combinations of options, and to display better print job dialogs (gray-out potentially conflicting options, highlight conflicting options that have been selected, display an error message when invalid comb9inatoins are submitted, etc.).

The constraints returned should should allow some level of “boolean logic”, including negation, to simplify the information returned. For example, to not allow doing finishing when transparencies are selected as the medium, it would be preferable if the constraints could express "(type – transparency) AND (finishing NOT = none)" instead of having to list a combination of "(type = transparency)" with every possible finishing value other than "none".

9.2.5 Support returning display hints

The API should support a means of returning "display hints". This is information that the application can use to display print options in a print dialog that is easy to use. For example, returning information about which options should be displayed on the "main window", which should be displayed in an "advanced" dialog, and which should not be displayed at all.

9.2.6 Support logically grouping features

The API should support a means of returning logical groupings of printer features. This is information about combinations of lower-level features that can be displayed and selected as a group to make the user interface easier to use. For example, a group of features called "black-and-white-draft" could include a logical setting of the color, resolution, and print density options.

The feature group support should be an open, extendible way for printer vendors and print administrators to express logical and commonly used groupings of print options that make it easier for end-users to take advantage of lower-level printer features. They should not be used to blindly list all possible combinations of a set of options, whether or not all the combinations make sense.

9.3 Interfaces

9.3.1 Query Functionality

The API used by the application to retrieve printer capabilities is the papiPrinterQuery function. See the description of that function for further details.

9.3.2 Capability Attributes

In addition to the xxx-supported attributes defined by the IPP standard [RFC2911], this section defines new attributes needed to satisfy the objectives described above.

9.3.2.1 Job-constraints-col (1 setOf collection)

Constraints are expressed in the printer object's job-constraints-col attribute. This attribute is multi-valued with each value having collection syntax. Each value is, in fact, an attribute list that represents one combination of job attributes/values which are not allowed for that printer. If an attribute in the collection does not have a value, then all values of that attribute are disallowed in this combination.

The set of values associated with job-constraints-col represents the complete set of job attribute constraints associated wit the containing printer object.

The attribute values in job-constraints-col may also be in range syntax, if the corresponding job attribute has integer syntax. This represents the included (or excluded, if the attribute is named in job-constraints-inverted) range of values of that attribute within that constraint.

9.3.2.2 Job-constraints-inverted (1setOf type2 keyword)

The job-constraints-inverted attribute lists the names of other attributes in the current job-constraints-col value whose comparison logic must be inverted. That is, the values of named attributes are to be excluded ("not equal to" values) from the constraint. If an attribute name is not included in the job-constraints-inverted attribute, then that attribute's values are to be included ("equal to" values) in the constraint.

You can think of the each att5ribute in a job-constraints-cols value as AND-ed together to express a disallowed combination of options: "(attr1 == values) AND (attr2 == values) AND ...". The job constraints-inverted attribute lists those attribute/value comparisons which are to be "!=" instead of "==".

9.3.3 Example

Here is an example of how the job-constraints-col attribute can be used to express various printer constraints. The example is expressed in pseudo-code with curly brackets enclosing each collection value and attributes within each collection are shown on separate lines with commas separating the values (this is the PAPI text encoding format documented in Chapter11: Attribute List Text Representation, with the addition of not-legal-syntax comments in "/* ... */" to help describe the examples):

```
job-constraints-col = {
        /*
         * Constraint: no high print quality with 240 dpi resolution
         * (print-quality == high) AND (printer-resolution == 240dpi)
         /
        {
                print-quality = high
                printer -resolution = 240dpi
        },

        /*
         * Constraint: no transparency with duplex
         * (sides != one-sided) AND (media – transparency)
         /
        {
                job-constraints-inverted = sides
                sides = one-sided
                media = transparency
        },

        /*
         * Constraint: no finishing with heavy-stock media
         * (finishings != none) AND (media == heavy-stock)
         /
        {
                job-constraints-inverted = finishing
                finishings = none
                media = heavy-stock
        },

        /*
         * Constraint: no duplex printing of A4 paper in landscape
         * (sides != one-sided) AND (media == A4) AND
         * (orientation-requested == landscape)
         /
        {
                job-constraints-inverted = sides
                sides = one-sided
                media = A4
                orientation-requested = landscape
        },

        /*
         * Constraint: no duplex printing of COM-10 envelopes
         * (sides != one-sided) AND (media == envelope) AND
         * (media-size == com10)
         /
        {
                job-constraints-inverted = sides
                sides = one-sided
                media = envelope
                media-size = com10
        },

        /*
         * Constraint: no stapling of greater than 50 sheets
         * (finishings == staple) AND (job-media-sheets > 50)
         */
        {
                job-constraints-inverted = job-media-sheets
                finishings = staple
                job-media-sheets = 1-50
        }
};
```

9.3.4 Validation Function

The API used by the application to validate print job attributes against printer capabilities is the papiJobValidate function. See the description of that function for further details.

Chapter 10: Attributes

For a summary of the IPP attributes which can be used with the PAPI interface, see: ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf

10.1 Extension Attributes

The following attributes are not currently defined by IPP, but may be used with this API.

10.1.1 Job-ticket-formats-supported

(1setOf type2 keyword) This optional printer attribute lists the job ticket formats that are supported by the printer. If this attribute is not present, it is assumed that the printer does not support any job ticket formats

10.1.2 media-margins

(1setOf integer) The media-margins attribute defines the printable margins for the current printer object and consists of exactly 4 or 8 ordered integers. Each group of 4 integers represent the minimum distance from the top, right, bottom, and left edges of the media in 100ths of millimeters.

If 4 integers are provided, the margins are the same for the front and back sides of the media when producing duplexed output. If 8 integers are provided, the first 4 integers represent the margins for the front side and the last 4 integers represent the margins for the back side of the media.

Currently the margin values only represent the minimum margins that can be used with all sizes and types of media. Future versions of the PAPI specification may define an interface for getting the margin values for specific combinations of job template attributes.

10.2 Required Job Attributes

The following job attributes must be supported to comply with this API standard. These attributes may be supported by the underlying print server directly, or they may be mapped by the PAPI library.

- job-id
- job-name
- job-originating-user-name
- job-printer-uri
- job-state
- job-state-reasons
- job-uri

- time-at-creation
- time-at-processing
- time-at-completed

10.3 Required Printer Attributes

The following printer attributes must be supported to comply with this API standard. These attributes may be supported by the underlying print server directly, or they may be mapped by the PAPI library.

- charset-configured
- charset-supported
- compression-supported
- document-format-default
- document-format-supported
- generated-natural-language-supported
- natural-language-configured
- operations-supported
- pdl-override-supported
- printer-is-accepting-jobs
- printer-name
- printer-state
- printer-state-reasons
- printer-up-time
- printer-uri-supported
- queued-job-count
- uri-authentication-supported
- uri-security-supported

10.4 IPP Attribute Type Mapping

The following table maps IPP to PAPI attribute value types:

IPP Type	***PAPI Type***
boolean	PAPI_BOOLEAN
charset	PAPI_STRING
collection	PAPI_COLLECTION
dateTime	PAPI_DATETIME
enum	PAPI_INTEGER (with C enum values)
integer	PAPI_INTEGER
keyword	PAPI_STRING
mimeMediaType	PAPI_STRING
name	PAPI_STRING
naturalLanguage	PAPI_STRING
octetString	not yet supported
rangeOfInteger	PAPI_RANGE
resolution	PAPI_RESOLUTION
text	PAPI_STRING
uri	PAPI_STRING
uriScheme	PAPI_STRING
1setOf X	C array
OOB (unused, delete, unsupported, etc.)	PAPI_METADATA (with enum value)

3358

Chapter 11: Attribute List Text Representation

11.1 ABNF Definition

The following ABNF definition [RFC2234] describes the syntax of PAPI attributes encoded as text options:

```
OPTION-STRING = [OPTION] *(1*WC OPTION) *WC

OPTION         = TRUEOPTION / FALSEOPTION / VALUEOPTION

TRUEOPTION     = NAME

FALSEOPTION    = "no" NAME

VALUEOPTION    = NAME "=" VALUE *( "," VALUE )

NAME           = 1*NAMECHAR

NAMECHAR       = DIGIT / ALPHA / "-" / "_" / "."

VALUE          = BOOLVALUE / COLVALUE / DATEVALUE / NUMBERVALUE /
QUOTEDVALUE /
                 RANGEVALUE / RESVALUE / STRINGVALUE

BOOLVALUE      = "yes" / "no" / "true" / "false"

COLVALUE       = "{" OPTION-STRING "}"

DATEVALUE      = HOUR MINUTE [ SECOND ] / YEAR MONTH DAY /
                 YEAR MONTH DAY HOUR MINUTE [ SECOND ]

YEAR           = 4DIGIT

MONTH          = "0" %x31-39 / "10" / "11" / "12"

DAY            = %x30-32 DIGIT / "1" DIGIT / "2" DIGIT / "30" / "31"

HOUR           = %x30-31 DIGIT / "1" DIGIT / "20" / "21" / "22" / "23"

MINUTE         = %x30-35 DIGIT

SECOND         = %x30-35 DIGIT

NUMBERVALUE    = 1*DIGIT / "-" 1*DIGIT / "+" 1*DIGIT

QUOTEDVALUE    = DQUOTE *QUOTEDCHAR DQUOTE / SQUOTE *QUOTEDCHAR SQUOTE

QUOTEDCHAR     = %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
                 %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
                                  %x5D-7E / %xA0-FF
```

```

OCTALDIGIT    = %x30-37

RANGEVALUE    = 1*DIGIT "-" 1*DIGIT

RESVALUE      = 1*DIGIT [ "x" 1*DIGIT ] ("dpi" / "dpc")

STRINGVALUE   = 1*STRINGCHAR

STRINGCHAR    = %x5C %x20 / %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
                %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
                                 %x5D-7E / %xA0-FF

SQUOTE        = %x27

WC            = %x09 / %x0A / %x20
```

11.2 Examples

The following example strings illustrate the format of text options:

11.2.1 Boolean Attributes

```
foo
nofoo
foo=false
foo=true
foo=no
foo=yes

```

11.2.2 Collection Attributes

```
media-col={media-size={x-dimension=123 y-dimension=456}}

```

11.2.3 Integer Attributes

```
copies=123
hue=-123

```

11.2.4 String Attributes

```
job-sheets=standard
job-sheets=standard,standard
media=na-custom-foo.8000-10000
job-name=John\'s\ Really\040Nice\ Document
```

11.2.5 String Attributes (quoted)

```
job-name="John\'s Really Nice Document"
document-name='Another \"Word\042 document.doc'
```

11.2.6 Range Attributes

```
page-ranges=1-5
page-ranges=1-2,5-6,101-120
```

11.2.7 Date Attributes

```
job-hold-until-datetime=1234
job-hold-until-datetime=123456
job-hold-until-datetime=20020904
job-hold-until-datetime=200209041234
job-hold-until-datetime=20020904123456
```

11.2.8 Resolution Attributes

```
resolution=360dpi
resolution=720x360dpi
resolution=1000dpc
```

11.2.9 Multiple Attributes

```
job-sheets=standard page-ranges=1-2,5-6,101-120 resolution=360dpi
```

Chapter 12: Conformance

There are some cases where it may not be necessary or even desirable to implement the interfaces defined in this specification in their entirety. This section describes which elements of the interfaces must be implemented and defines sets of interfaces that may be implemented. The sets of interfaces that may be implemented define various levels of conformance. Conformance to a particular level may only be claimed by an implementation if and only if all of the interfaces defined in that level are implemented as described in their associated section of this document. These implementations may only return PAPI_OPERATION_NOT_SUPPORTED if and only if the underlying support has been administratively disabled. Regardless of conformance level claimed by an implementation, the header file for every implementation must be complete. That is to say that it must include a complete set of type definitions, enumeration and function prototypes.

12.1 Query Profile

The Query Profile is defined to provide querying functionality. A PAPI implementation conforming to the Query Profile must provide code for all functions defined in the PAPI and must support all of the definitions in the "papi.h" C header file. For each function defined in the PAPI specification, a conforming implementation must either perform the requested function or return the PAPI_OPERATION_NOT_SUPPORTED status code (see section 3.8). The PAPI_OPERATION_NOT_SUPPORTED status code indicates either (1) that the PAPI implementation doesn't provide any support for the function, i.e., the function is stubbed out, or (2), the PAPI implementation does provide *code support* for the function, but the Printer or Print system selected by the application does not support the corresponding function.

lists the functions and attributes that a PAPI implementation is REQUIRED to provide *code support* in order to claim conformance to the Query Profile. The blank entries are OPTIONAL for a PAPI implementation to support.

12.2 Job Submission Profile

The Job Submission Profile is defined to provide the job submission functionality and is a superset of the Querying Profile. lists the functions and attributes that a PAPI implementation is REQUIRED to provide *code support* in order to claim conformance to the Job Submission Profile. The blank entries are OPTIONAL for a PAPI implementation to support.

12.3 Conformance Table

PAPI Functions & Attributes	***Query Profile***	***Job Submission Profile***
Chapter3: Common Structures	All Structures	All Structures

PAPI Functions & Attributes	***Query Profile***	***Job Submission Profile***
Chapter4: Attributes API		
4.1 papiAttributeListAdd	REQUIRED	REQUIRED
4.2 papiAttributeListAddString	REQUIRED	REQUIRED
4.3 papiAttributeListAddInteger	REQUIRED	REQUIRED
4.4 papiAttributeListAddBoolean	REQUIRED	REQUIRED
4.5 papiAttributeListAddRange	REQUIRED	REQUIRED
4.6 papiAttributeListAddResolution	REQUIRED	REQUIRED
4.7 papiAttributeListAddDatetime	REQUIRED	REQUIRED
4.8 papiAttributeListAddCollection	REQUIRED	REQUIRED
4.9 papiAttributeListDelete	REQUIRED	REQUIRED
4.10 papiAttributeListGetValue	REQUIRED	REQUIRED
4.11 papiAttributeListGetString	REQUIRED	REQUIRED
4.12 papiAttributeListGetInteger	REQUIRED	REQUIRED
4.13 papiAttributeListGetBoolean	REQUIRED	REQUIRED
4.14 papiAttributeListGetRange	REQUIRED	REQUIRED
4.15 papiAttributeListGetResolution	REQUIRED	REQUIRED
4.16 papiAttributeListGetDatetime	REQUIRED	REQUIRED
4.17 papiAttributeListGetCollection	REQUIRED	REQUIRED
4.18 papiAttributeListFree	REQUIRED	REQUIRED
4.19 papiAttributeListFind	REQUIRED	REQUIRED
4.20 papiAttributeListGetNext	REQUIRED	REQUIRED
4.21 papiAttributeListFromString		
4.22 papiAttributeListToString		
Chapter5: Service API	All Functions	All Functions
Chapter6: Printer API		
6.2 papiPrintersList	REQUIRED	REQUIRED
6.3 papiPrinterQuery	REQUIRED	REQUIRED
6.4 papiPrinterModify		
6.5 papiPrinterPause		

PAPI Functions & Attributes	***Query Profile***	***Job Submission Profile***
6.6 papiPrinterResume		
6.7 papiPrinterPurgeJobs		
6.8 papiPrinterListJobs	REQUIRED	REQUIRED
6.9 papiPrinterGetAttributeList	REQUIRED	REQUIRED
6.10 papiPrinterFree	REQUIRED	REQUIRED
6.11 papiPrinterListFree	REQUIRED	REQUIRED
Chapter7: Job API		
7.1 papiJobSubmit		REQUIRED
7.2 papiJobSubmitByReference		REQUIRED
7.3 papiJobValidate		
7.4 papiJobStreamOpen		REQUIRED
7.5 papiJobStreamWrite		REQUIRED
7.6 papiJobStreamClose		REQUIRED
7.7 papiJobQuery	REQUIRED	REQUIRED
7.8 papiJobModify		REQUIRED
7.9 papiJobCancel		REQUIRED
7.10 papiJobHold		REQUIRED
7.11 papiJobRelease		REQUIRED
7.12 papiJobRestart		REQUIRED
7.13 papiJobGetAttributeList		REQUIRED
7.14 papiJobGetPrinterName		REQUIRED
7.15 papiJobGetId		REQUIRED
7.16 papiJobGetJobTicket		
7.17 papiJobFree		REQUIRED
7.18 papiJobListFree		REQUIRED
Chapter8: Miscellaneous API		
8.1 papiStatusString		REQUIRED
8.2 papiLibrarySupportedCalls		REQUIRED
8.3 papiLibrarySupportedCall		REQUIRED

PAPI Functions & Attributes	***Query Profile***	***Job Submission Profile***
Chapter9: Attributes		
9.1.1 Job-ticket-formats-supported	REQUIRED	REQUIRED
9.1.2 media-margins	REQUIRED	REQUIRED
9.2 Required Job Attributes	REQUIRED	REQUIRED
9.3 Required Printer Attributes	REQUIRED	REQUIRED
9.4 IPP Attribute Type Mapping	REQUIRED	REQUIRED

Chapter 13: Sample Code

Sample implementations of this specification and client applications built upon it can be found at http://sf.net/projects/OpenPrinting/ While the implemenations and clients applications found there are intended to be true to the spec, they are not authoritative. This document is the athoritavie definition of the Free Standard Group Open Standard Print API (PAPI).

Chapter 14: References

14.1 Internet Printing Protocol (IPP)

IETF RFCs can be obtained from "http://www.rfc-editor.org/rfcsearch.html". Other IPP documents can be obtained from "http://www.pwg.org/ipp/index.html" and "ftp://ftp.pwg.org/pub/pwg/ipp/new_XXX/".

[RFC2911] T. Hastings R. Herriot R. deBry S. Isaacson and P. Powell August 1998 Internet Printing Protocol/1.1: Model and Semantics (Obsoletes 2566)

[RFC3196] T. Hastings H. Holst C. Kugler C. Manros and P. Zehler November 2001 Internet Printing Protocol/1.1: Implementor's Guide

[RFC3380] T. Hastings R. Herriot C. Kugler and H. Lewis September 2002 Internet Printing Protocol (IPP): Job and Printer Set Operations

[RFC3381] T. Hastings H. Lewis and R. Bergman September 2002 Internet Printing Protocol (IPP): Job Progress Attributes

[RFC3382] R. deBry T. Hastings R. Herriot K. Ocke and P. Zehler September 2002 Internet Printing Protocol (IPP): The 'collection' attribute syntax

[5100.2] T. Hastings and R. Bergman IEEE-ISTO 5100.2 February 2001 Internet Printing Protocol (IPP): output-bin attribute extension

[5100.3] T. Hastings and K. Ocke IEEE-ISTO 5100.3 February 2001 Internet Printing Protocol (IPP): Production Printing Attributes

[5100.4] R. Herriot and K. Ocke IEEE-ISTO 5100.4 February 2001 Internet Printing Protocol (IPP): Override Attributes for Documents and Pages

[5101.1] T. Hastings and D. Fullman IEEE-ISTO 5101.1 February 2001 Internet Printing Protocol (IPP): finishings attribute values extension

[ops-set2] C. Kugler T. Hastings and H. Lewis July 2001 Internet Printing Protocol (IPP): Job and Printer Administrative Operations

14.2 Job Ticket

[jdf] CIP4 Organization April 2002 Job Definition Format (JDF) Specification Version 1.1

14.3 Printer Working Group (PWG)

[PWGSemMod] P. Zehler and Albright September 2002 Printer Working Group (PWG): Semantic Model

14.4 Other

[RFC1738] T. Berners-Lee L. Masinter and M. McCahill December 1994 Uniform Resource Locators (URL) (Updated by RFC1808, RFC2368, RFC2396)

[RFC2234] D. Crocker and P. Overell November 1997 Augmented BNF for Syntax Specifications: ABNF

[RFC2396] T. Berners-Lee R. Fielding and L. Masinter August 1998 Uniform Resource Locators (URL): Generic Syntax (Updates RFC1808, RFC1738)

Chapter 15: Change History

15.1 Version 1.0 (May 9, 2005)

Added note about interfaces to be covered in the next release.

15.2 Version 0.92 (January 12, 2005).

Added administrative operations: papiPrinterAdd, papiPrinterRemove, papiPrinter Enable, papaPrinterDisable, papiJobPromote.

15.3 Version 0.91 (January 28, 2004).

Pruned several example code excerpts to the essential information required to get a better understanding of the various calls.

Added/modified introductory text for Attribute, Service, Printer, and Job API chapters.

Added papi_metadata_t type/support for various OOB IPP types that we need to support.

Converted from SGML to OpenOffice to be able to use versioning, change bars, line number, ... (will begin using versioning and change bars after this release)

Added numerous cross references.

Added papiLibrarySupportedCall() and papiLibrarySupportedCalls(). To enumerate/verify actual support for a function in the library

Added papiServiceGetAttributeList() call to retrieve print service and implementation specific information from a service handle.

Added a “Conformance” section to the document. A draft introduction and conformance table are included, but the actual conformance levels need work. The bulk of this was included from Ira's and Tom’s draft.

Moved Attribute section in front of the Service, Printer, and Job sections interfaces to improve flow of document.

Added papi_encryption_t to common structures

Added constraints chapter. The bulk of this chapter was copied directly from v0.3 of the papi capabilities document.

15.4 Version 0.9 (November 18, 2002).

Changed media-margins order to "top, right, bottom, left" to match other standards.

Changed media-margins units to "100ths of millimeters" to match other standards. Also, reworded last paragraph of description of this attribute.

15.5 Version 0.8 (November 15, 2002).

Added value field, explanation, and corrected example for papi_filter_t.

Added media-margins attribute to "Extension Attributes" section.

Renamed function names with "Username" to "UserName", and renamed function names with "Servicename" to "ServiceName", and Miscellaneous wording and typo corrections.

15.6 Version 0.7 (October 18, 2002).

Added attr_delim argument to papiAttributeListToString and made new-line ("\n") an allowed attribute delimiter on input to papiAttributeListFromString.

Added "Semantics Reference" subsections to functions.

Added to References: [5101.1], [RFC3196], and URIs for obtaining IPP documents.

Added PAPI_JOB_TICKET_NOT_SUPPORTED status code.

Added "Globalization" section in the "Print System Model" chapter.

Changed definition and usage of returned value from papiAttributeListGetValue. Also clarified what happens to output values when a papiAttributeListGet* call has an error.

Clarified descriptions of papiPrinterGetAttributeList and papiJobGetAttributeList.

Changed buffer length arguments from int to size_t.

Clarified that papiServiceDestroy must always be called after a call to papiServiceCreate.

Removed attributes-charset, attributes-natural-language, and job-printer-up-time from the "Required Job Attributes" (they may be hidden inside the PAPI implementation).

Clarified result of passing both attributes and a job ticket on all the job submission functions.

Miscellaneous wording and typo corrections.

15.7 Version 0.6 (September 20, 2002)

Made explanation of requested_attrs in papiPrintersList the same as it is for papiPrinterQuery.

Moved units argument on papiAttributeListAddResolution to the end of the argument list to match the corresponding get function.

Added papiAttributeListAddCollection and papiAttributeListGetCollection.

Removed unneeded extra level of indirection from attrs argument to papiAttributeListGet* functions. Also made the attrs argument const.

Added note to "Conventions" section that strings are assumed to be UTF-8 encoded.

Added papiAttributeListFromString and papiAttributeListToString functions, along with a new appendix defining the string format syntax.

Added papiJobSubmitByReference, papiJobStreamOpen, papiJobStreamWrite, and papiJobStreamClose functions.

Added short "Document" section in the "Print System Model" chapter.

Added explanation of how multiple files specified in the papiJobSubmit file_names argument are handled by the print system.

Changed papi_job_ticket_t "uri" field to "file_name" and added explanation text.

Added explanation of implementation option for merging papiJobSubmit attributes with job_ticket argument.

Added "References" appendix.

Added "IPP Attribute Type Mapping" appendix.

Added "PWG" job ticket format to papi_jt_format_t.

Miscellaneous wording and typo corrections.

15.8 Version 0.5 (August 30, 2002).

Added job_attrs argument to papiPrinterQuery to support more accurate query of printer capabilities.

Added management functions papiAttributeDelete, papiJobModify, and papiPrinterModify.

Added functions papiAttributeListGetValue, papiAttributeListGetString, papiAttributeListGetInteger, etc.

Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent with the naming convention (first word after "papi" is the object being operated upon).

Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.

Made description of authentication more implementation-independent.

Added reference to IPP attributes summary document.

Added result argument to papiPrinterPurgeJobs.

Added "collection attribute" support (PAPI_COLLECTION type).

Changed boolean values to consistently use char. Added PAPI_FALSE and PAPI_TRUE enum values.

15.9 Version 0.4 (July 19, 2002).

Made papi_job_t and papi_printer_t opaque handles and added "get" functions to access the associated information (papiPrinterGetAttributeList, papiJobGetAttributeList, papiJobGetId, papiJobGetPrinterName, papiJobGetJobTicket).

Removed variable length argument lists from attribute add functions.

Changed order and name of flag value passed to attribute add functions.

Eliminated indirection in date/time value passed to papiAttributeAddDatetime.

Added message argument to papiPrinterPause.

15.10 Version 0.3 (June 24, 2002).

Converted to DocBook format from Microsoft Word

Major rewrite, including:

Changed how printer names are described in "Model/Printer"

Changed fixed length strings to pointers in numerous structures/sections

Redefined attribute/value structures and associated API descriptions

Changed list/query functions to return "objects"

Rewrote "Attributes API" chapter

Changed many function definitions to pass NULL-terminated arrays of pointers instead of a separate count argument

Changed papiJobSubmit to take an attribute list structure as input instead of a formatted string

15.11 Version 0.2 (April 17, 2002).

Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)

Filled in "Encryption" section and added information about encryption in "Object Identification" section

Added "short_name" field in "Object Identification" section

Added "Job Ticket (papi_job_ticket_t)" section

Added papiPrinterPause

Added papiPrinterResume

Added papiPurgeJobs

Added optional job_ticket argument to papiJobSubmit

Added optional passing of filenames by URI to papiJobSubmit

Added papiHoldJob

Added papiReleaseJob

Added papiRestartJob

15.12 Version 0.1 (April 3, 2002).

Original draft version