Open Standard Print API (PAPI)

Version 0.3 (DRAFT)

Alan Hlava
IBM Printing Systems Division

Norm Jacobs
Sun Microsystems, Inc.

Michael R Sweet
Easy Software Products

Open Standard Print API (PAPI): Version 0.3 (DRAFT)

by Alan Hlava, Norm Jacobs, and Michael R Sweet

Version 0.3 (DRAFT) Edition

Table of Contents

Chapter 1. Introduction

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

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

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

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

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

Printer	Queue	Print Resource (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 are the semantics defined by the Internet Print Protocol (IPP) standard. This is a fairly simple model in terms of the number of object types. It is defined very clearly and in detail in the IPP RFC 2911, Chapter 2 (http://ietf.org/rfc/rfc2911.txt?number=2911).

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

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

2.2.1. Print Service

PAPI includes the concept of a "Print Service". This is the entity which the PAPI interface communicates with in order to actually perform the requested print operations. The print service may be a remote print server, a local print server, an "intelligent" printer, etc.

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 via 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"). The PAPI implementation may detect and map short names to long global names in an implementation-specific way.

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, and they 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.

2.3. Security

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

2.3.1. Authentication

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

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

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

2.3.2. Authorization

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

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

2.3.3. Encryption

Encrypting certain data sent to and from the print service may be desirable in some environments. See field "encryption" in Section 3.2 for how to request encryption on a print operation. Note that some print services may not support encryption. To comply with this standard, only the HTTP_ENCRYPT_NEVER value must be supported.

Chapter 3. Common Structures

3.1. Conventions

- All "char*" variables and fields are pointers to standard C/C++ NULL-terminated 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 contain information about the print service which is being used to handle the PAPI requests. It is typically created once, used on one or more subsequent PAPI calls, and then deleted.

```
typedef void* papi_service_t;
```

Included in the information associated with a papi_service_t is a definition about how requests whould be encrypted.

```
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 HTTP_ENCRYPT_NEVER value must be supported.

3.3. Attributes and Values

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_attribute_value_type_t;
```

* *ISSUE: Are other types needed to support the newer IPP "collection" attrs?*

```
/* 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;
    } resolution;

    time_t datetime;  /* PAPI_DATETIME value */
} papi_attribute_value_t;
```

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

```
/* Attribute update types */
#define PAPI_ATTR_APPEND  0x0001 /* Add values to attr  */
#define PAPI_ATTR_REPLACE 0x0002 /* Delete existing
                               values then add new ones */
#define PAPI_ATTR_EXCL    0x0004 /* Fail if attr exists */
```

For the valid attribute names which may be supported, see Chapter 9.

3.4. Job Object (papi_job_t)

This structure represents a job object.

```
typedef struct
{
    char* name;
    int32_t id;
    papi_attribute_t** attributes;
    papi_job_ticket_t* job_ticket;
} papi_job_t;

```

The "name" field contains the printer name or URI.

The "id" field contains the local job identification number. This number is only unique in the context of a particular printer.

The "attributes" field points to an attribute list associated with the job.

The "job_ticket" field points to a structure representing the job's associated job ticket. A NULL value indicates there is no associated job ticket.

3.5. Printer Object (papi_printer_t)

This structure represents a printer object.

```
typedef struct
{
    char* name;
    papi_attribute_t** attributes;
} papi_printer_t;

```

The "name" field contains the printer name or URI.

The "attributes" field points to an attribute list associated with the printer.

3.6. Job Ticket (papi_job_ticket_t)

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

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

* *ISSUE: What other formats are needed in the above?*

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

* *ISSUE: Need general statement about JT vs. attribute precedence here*

3.7. 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_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.

3.8. 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
    {
        unsigned int  mask; /* PAPI_FILTER_BITMASK */

        /* future filter types may be added here */
    } u;
} papi_filter_t;
```

For papiPrintersList requests, the following values may be OR-ed together and used in the papi_filter_t mask field to limit the printers returned.

```
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) */
  };
```

* *ISSUE: Do all of the above apply in PAPI?*

Chapter 4. Service API

4.1. papiServiceCreate

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.

Syntax.

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

Inputs.

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.; this is not addressed by this standard).

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.

password

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

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 cancelled and non-zero if new authentication information has been set.

encryption

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

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.

Outputs.

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 which case it may be set to NULL.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp:/printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
...
status = papiServiceCreate(&handle,
                           service_name,
                           user_name,
                           password,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiServiceCreate failed: %s\n",
            papiStatusString(status));
    if (handle != NULL)
    {
        fprintf(stderr, "    details: %s\n",
            papiServiceGetStatusMessage(handle));
    }
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUsername, papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB

4.2. papiServiceDestroy

Description. Destroy a print service handle and free the resources associated with it. If there is application data associated with the service handle, it is the caller's responsibility to free this memory.

Syntax.

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

Inputs.

handle

The print service handle to be destroyed.

Outputs. none

Returns. none

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp://printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
...
status = papiServiceCreate(&handle,
                           service_name,
                           user_name,
                           password,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate

4.3. papiServiceSetUsername

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.

Syntax.

```
papi_status_t papiServiceSetUsername(
        papi_service_t handle,
        const char* user_name );
```

Inputs.

handle

Handle to the print service to update.

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.

Outputs. handle is updated.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* user_name = "pappy";
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiServiceSetUsername(handle, user_name);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiServiceSetUsername failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

4.4. papiServiceSetPassword

Description. Set the user 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.

Syntax.

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

Inputs.

handle

Handle to the print service to update.

password

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

Outputs. handle is updated.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* password = "goober";
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiServiceSetPassword(handle, password);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiServiceSetPassword failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceSetUsername, papiServiceGetStatusMessage

4.5. papiServiceSetEncryption

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

Syntax.

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

```

Inputs.

handle

Handle to the print service to update.

encryption

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

Outputs. handle is updated.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
if (status != PAPI_OK)
```

```
{
    /* handle the error */
    fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceGetStatusMessage

4.6. papiServiceSetAuthCB

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

Syntax.

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

Inputs.

handle

Handle to the print service to update.

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 cancelled and non-zero if new authentication information has been set.

Outputs. handle is updated.

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

Example.

```
#include "papi.h"

extern int get_password(papi_service_t handle);
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiServiceSetAuthCB(handle, get_password);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiServiceSetAuthCB failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceGetStatusMessage

4.7. papiServiceSetAppData

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.

Syntax.

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

Inputs.

handle

Handle to the print service to update.

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.

Outputs. handle is updated.

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

Example.

```
#include "papi.h"

extern int get_password(papi_service_t handle);
papi_status_t status;
papi_service_t handle = NULL;
char* app_data = "some data";
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiServiceSetAppData(handle, app_data);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiServiceSetAppData failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceGetStatusMessage

4.8. papiServiceGetServicename

Description. Get the service name associated with the print service handle.

Syntax.

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

Inputs.

handle

Handle to the print service.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
char* service_name = NULL;
...
service_name = papiServiceGetServicename(handle);
if (service_name != NULL)
{
    /* use the returned name */
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate

4.9. papiServiceGetUsername

Description. Get the user name associated with the print service handle.

Syntax.

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

Inputs.

handle

Handle to the print service.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
char* user_name = NULL;
...
user_name = papiServiceGetUsername(handle);
if (user_name != NULL)
{
    /* use the returned name */
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceSetUsername

4.10. papiServiceGetPassword

Description. Get the user password associated with the print service handle.

Syntax.

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

Inputs.

handle

Handle to the print service.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
char* password = NULL;
...
password = papiServiceGetPassword(handle);
if (password != NULL)
{
    /* use the returned password */
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceSetPassword

4.11. papiServiceGetEncryption

Description. Get the type of encryption associated with the print service handle.

Syntax.

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

Inputs.

handle

Handle to the print service.

Outputs. none

Returns. The type of encryption associated with the print service handle.

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
papi_encryption_t encryption;
...
encryption = papiServiceGetEncryption(handle);
/* use the returned encryption value */
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceSetEncryption

4.12. papiServiceGetAppData

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

Syntax.

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

Inputs.

handle

Handle to the print service.

Outputs. none

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

Example.

```
#include "papi.h"
```

```
papi_status_t status;
papi_service_t handle = NULL;
char* app_data = NULL;
...
app_data = (char*)papiServiceGetAppData(handle);
if (app_data != NULL)
{
    /* use the returned application data */
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiServiceCreate, papiServiceSetAppData

4.13. papiServiceGetStatusMessage

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 submittor of the original operation.

Syntax.

```
const char* papiServiceGetStatusMessage(
        const papi_service_t handle );
```

Inputs.

handle

Handle to the print service.

Outputs. none

Returns. Pointer to the message associated with the status of the last operation performed.

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* user_name = "pappy";
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiServiceSetUsername(handle, user_name);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiServiceSetUsername failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}
...
papiServiceDestroy(handle);
```

See Also. papiStatusString

Chapter 5. Printer API

5.1. Usage

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. The using program would 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). The using program would then call papiPrinterListFree to free the returned results.

5.2. papiPrintersList

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.

Syntax.

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

Inputs.

handle

Handle to the print service to use.

requested_attrs

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

filter

(optional) Pointer to a filter to limit the number if printers returned on the list request. See Section 3.8 for details. If NULL is passed then all known printers are listed.

Outputs.

printers

List of printer objects that matched the filter criteria.

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

Example.

```
#include "papi.h"

int i;
papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp://printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
const char* req_attrs[] =
{
    "printer-name",
    "printer-location",
    NULL
};
const papi_filter_t filter =
      PAPI_PRINTER_BW | PAPI_PRINTER_DUPLEX;
papi_printer_t** printers = NULL;
...
status = papiServiceCreate(&handle,
                           service_name,
                           user_name,
                           password,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}
```

```
status = papiPrinterList(handle,
                         req_attrs,
                         filter,
                         &printers);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiPrinterList failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}

if (printers != NULL)
{
    for (i=0; printers[i] != NULL; i++)
    {
        /* process the printer object */
        ...
    }
    papiPrinterListFree(printers);
}

papiServiceDestroy(handle);
```

See Also. papiPrinterListFree, papiPrinterQuery

5.3. papiPrinterQuery

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

name

The name or URI of the printer to query.

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.)

Outputs.

printer

Pointer to a printer object containing the requested attributes.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp://printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
const char* printer_name = "my-printer";
const char* req_attrs[] =
{
    "printer-name",
    "printer-location",
    "printer-state",
    "printer-state-reasons",
    "printer-state-message",
    NULL
};
papi_printer_t* printer = NULL;
...
status = papiServiceCreate(&handle,
                           service_name,
                           user_name,
                           password,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
```

```
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

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

papiServiceDestroy(handle);
```

See Also. papiPrinterList, papiPrinterFree

5.4. papiPrinterPause

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.

Depending on the implementation, this function may also stop the print service from processing currently printing job(s).

Syntax.

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

Inputs.

handle

Handle to the print service to use.

name

The name or URI of the printer to operate on.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp://printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
const char* printer_name = "my-printer";
...
status = papiServiceCreate(&handle,
                           service_name,
                           user_name,
                           password,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

See Also. papiPrinterResume

5.5. papiPrinterResume

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.

Syntax.

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

Inputs.

handle

    Handle to the print service to use.

name

    The name or URI of the printer to operate on.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp://printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
const char* printer_name = "my-printer";
...
status = papiServiceCreate(&handle,
                           service_name,
                           user_name,
                           password,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);
if (status != PAPI_OK)
```

```
{
    /* handle the error */
    ...
}

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

papiServiceDestroy(handle);
```

See Also. papiPrinterPause

5.6. papiPrinterPurgeJobs

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

name

The name or URI of the printer to operate on.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* service_name = "ipp://printserv:631";
const char* user_name = "pappy";
const char* password = "goober";
const char* printer_name = "my-printer";
...
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);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}

papiServiceDestroy(handle);
```

See Also. papiJobCancel

5.7. papiPrinterListJobs

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

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.)

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 assoc 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 */
```

max_num_jobs

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

Outputs.

jobs

List of job objects returned.

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

Example.

```
#include "papi.h"

int i;
papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_printer_t** printers = NULL;
const char* job_attrs[] =
{
    "job-id",
    "job-name",
    "job-originating-user-name",
    "job-state",
    "job-state-reasons",
    NULL
};
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiPrinterListJobs(handle,
                            printer_name,
                            job_attrs,
                            PAPI_LIST_JOBS_ALL,
                            0,
                            &jobs);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiPrinterListJobs failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}

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

papiServiceDestroy(handle);
```

See Also. papiJobQuery, papiJobListFree

5.8. papiPrinterFree

Description. Free a printer object.

Syntax.

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

Inputs.

printer

Pointer to the printer object to free.

Outputs. none

Returns. none

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_printer_t* printer = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
```

```
        ...
    }

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

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

    papiServiceDestroy(handle);
```

See Also. papiPrinterQuery

5.9. papiPrinterListFree

Description. Free a list of printer objects.

Syntax.

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

Inputs.

printers

    Pointer to the printer object list to free.

Outputs. none

Returns. none

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_printer_t** printers = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

if (printers != NULL)
{
    /* process the printer objects */
    ...
    papiPrinterListFree(printers);
}

papiServiceDestroy(handle);
```

See Also. papiPrinterList

Chapter 6. Attributes API

6.1. papiAttributeAdd

Description. Add an attribute/value to an attribute list. 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 papiAttributeAddString, papiAttributeAddInteger, etc. functions defined later in this chapter.

Syntax.

```
papi_status_t papiAttributeAdd(
          papi_attribute_t*** attrs,
          const char* name,
          const papi_attribute_value_type_t type,
          const int update_type,
          const int num_values,
          ... );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

type

The type of values for this attribute.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

num_values

The number of values that follow in the variable part of the argument list.

...

The values to be added.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAdd(&attrs,
                 "job-name",
                 PAPI_STRING,
                 PAPI_EXCL,
                 1,
                 "My job" );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree, papiAttributeAddString, papiAttributeAddInteger, papiAttributeAddBoolean, papiAttributeAddRange, papiAttributeAddResolution, papiAttributeAddDatetime

6.2. papiAttributeAddString

Description. Add a string-valued attribute to an attribute list. 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.

Syntax.

```
papi_status_t papiAttributeAddString(
        papi_attribute_t*** attrs,
        const char* name,
        const int update_type,
        const int num_values,
        ... );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

num_values

The number of values that follow in the variable part of the argument list.

...

The values (char*) to be added.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAddString(&attrs,
                       "job-name",
                       PAPI_EXCL,
                       1,
                       "My job" );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree, papiAttributeAdd

6.3. papiAttributeAddInteger

Description. Add an integer-valued attribute to an attribute list. 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.

Syntax.

```
papi_status_t papiAttributeAddInteger(
          papi_attribute_t*** attrs,
          const char* name,
          const int update_type,
          const int num_values,
          ... );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

num_values

The number of values that follow in the variable part of the argument list.

...

The values (int) to be added.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAddInteger(&attrs,
                        "copies",
                        PAPI_EXCL,
                        1,
                        3 );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree, papiAttributeAdd

6.4. papiAttributeAddBoolean

Description. Add a boolean-valued attribute to an attribute list. 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.

Syntax.

```
papi_status_t papiAttributeAddBoolean(
         papi_attribute_t*** attrs,
         const char* name,
         const int update_type,
         const int num_values,
         ... );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

num_values

The number of values that follow in the variable part of the argument list.

...

The values (0 or 1) to be added.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAddBoolean(&attrs,
                        "color-supported",
                        PAPI_EXCL,
                        1,
                        1 );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree, papiAttributeAdd

6.5. papiAttributeAddRange

Description. Add a range-valued attribute to an attribute list. 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.

Syntax.

```
papi_status_t papiAttributeAddRange(
          papi_attribute_t*** attrs,
          const char* name,
          const int update_type,
          const int lower,
          const int upper );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

lower

The lower range value.

upper

The upper range value.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAddRange(&attrs,
                       "job-k-octets-supported",
                       PAPI_EXCL,
                       1,
                       100000 );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree

6.6. papiAttributeAddResolution

Description. Add a resolution-valued attribute to an attribute list. 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.

Syntax.

```
papi_status_t papiAttributeAddRange(
        papi_attribute_t*** attrs,
        const char* name,
        const int update_type,
        const int xres,
        const int yres );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

xres

The X-axis resolution value.

yres

The Y-axis resolution value.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAddResolution(&attrs,
                        "printer-resolution",
                        PAPI_EXCL,
                        300,
                        300 );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree

6.7. papiAttributeAddDatetime

Description. Add a date/time-valued attribute to an attribute list. 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.

Syntax.

```
papi_status_t papiAttributeAddDatetime(
          papi_attribute_t*** attrs,
          const char* name,
          const int update_type,
          const time_t* date_time );
```

Inputs.

attrs

Points to an attribute list. If a NULL value is passed, this function will allocate the attribute list.

name

Points to the name of the attribute to add.

update_type

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that indicates how to handle the request if the attribute already exists in the attribute list.

date_time

The date/time value.

Outputs.

attrs

The attribute list is updated.

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

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
time_t date_time
...
time(&date_time);
papiAttributeAddDatetime(&attrs,
                        "date-time-at-creation",
                        PAPI_EXCL,
                        &date_time );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFree

6.8. papiAttributeListFree

Description. Frees an attribute list.

Syntax.

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

Inputs.

attrs

Attribute list to be freed.

Outputs. none

Returns. none

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
...
papiAttributeAddString(&attrs,
                       "job-name",
                       PAPI_EXCL,
                       1,
                       "My job" );
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeAddString, etc.

6.9. papiAttributeListFind

Description. Find an attribute in an attribute list.

Syntax.

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

Inputs.

attrs

Attribute list to be searched.

name

Pointer to the name of the attribute to find.

Outputs. none

Returns. Pointer to the found attribute. NULL indicates that the specified attribute was not found

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
papi_attribute_t*  attr = NULL;
...
attr = papiAttributeListFind(&attrs,
                        "job-name" );
if (attr != NULL)
{
    /* process the attribute */
    ...
}
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListGetNext

6.10. papiAttributeListGetNext

Description. Get the next attribute in an attribute list.

Syntax.

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

Inputs.

attrs

Attribute list to be used.

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.

Outputs. none

Returns. Pointer to the found attribute. NULL indicates that the end of the attribute list was reached.

Example.

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
papi_attribute_t*  attr = NULL;
void* iterator = NULL;
...
attr = papiAttributeListGetNext(&attrs,
                       &iterator );
while (attr != NULL)
{
    /* process this attribute */
    ...
    attr = papiAttributeListGetNext(&attrs,
                       &iterator );
}
...
papiAttributeListFree(attrs);
```

See Also. papiAttributeListFind

Chapter 7. Job API

7.1. papiJobSubmit

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

printer_name

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

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.

job_ticket

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

file_names

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

Outputs.

job

The resulting job object representing the submitted job.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer = "my-printer";
const papi_attribute_t** attrs = NULL;
const papi_job_ticket_t* ticket = NULL;
const char* files[] = { "/etc/motd", NULL };
papi_job_t* job = NULL;

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

papiAttributeAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
                                 PAPI_STRING, 1, "test job");
papiAttributeAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
                                 PAPI_INTEGER, 1, 4);

status = papiJobSubmit(handle,
                       printer,
                       attrs,
                       ticket,
                       files,
                       &job);
if (status != PAPI_OK)
{
    fprintf(stderr, "papiJobSubmit failed: %s\n",
             papiStatusString(status));
    ...
}

if (job != NULL)
{
    /* look at the job object (maybe get the id) */

    papiJobFree(job);
}

papiServiceDestroy(handle);
```

See Also. papiJobValidate, papiJobFree

7.2. papiJobValidate

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

printer_name

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

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.

job_ticket

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

file_names

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

Outputs.

job

The resulting job object representing what would be the submitted job.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer = "my-printer";
const papi_attribute_t** attrs = NULL;
const papi_job_ticket_t* ticket = NULL;
const char* files[] = { "/etc/motd", NULL };
papi_job_t* job = NULL;

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

papiAttributeAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
                                 PAPI_STRING, 1, "test job");
papiAttributeAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
                                 PAPI_INTEGER, 1, 4);

status = papiJobValidate(handle,
                       printer,
                       attrs,
                       ticket,
                       files,
                       &job);
if (status != PAPI_OK)
{
    fprintf(stderr, "papiJobValidate failed: %s\n",
             papiStatusString(status));
    ...
}

if (job != NULL)
{
    ...
    papiJobFree(job);
}
```

```
papiServiceDestroy(handle);
```

See Also. papiJobSubmit, papiJobFree

7.3. papiJobQuery

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

printer_name

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

job_id

The ID number of the job to be queried.

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.)

Outputs.

job

The returned job object containing the requested attributes.

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_job_t* job = NULL;
int32_t job_id = 12;
const char* job_attrs[] =
{
    "job-id",
    "job-name",
    "job-originating-user-name",
    "job-state",
    "job-state-reasons",
    NULL
};
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

if (job != NULL)
{
    /* process the job */
    ...
    papiJobFree(job);
}
```

```
papiServiceDestroy(handle);
```

See Also. papiJobFree, papiPrinterListJobs

7.4. papiJobCancel

Description. Cancel the specified print job.

Syntax.

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

Inputs.

handle

Handle to the print service to use.

printer_name

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

job_id

The ID number of the job to be cancelled.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
int32_t job_id = 12;
...
```

```
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

papiServiceDestroy(handle);
```

See Also. papiPrinterListJobs, papiPrinterPurgeJobs

7.5. papiJobHold

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

Syntax.

```
papi_status_t papiJobHold(
                papi_service_t      handle,
         const char*                printer_name,
         const int32_t              job_id,
         const char*                hold_until,
         const time_t*              hold_until_time );
```

Inputs.

handle

Handle to the print service to use.

printer_name

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

job_id

The ID number of the job to be held.

hold_until

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

hold_until_time

(optional) Specifies the time when the job will be automatically released for printing. This argument is ignored unless "timed" is passed as the hold_until argument.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
int32_t job_id = 12;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

papiServiceDestroy(handle);
```

See Also. papiJobRelease

7.6. papiJobRelease

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

printer_name

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

job_id

The ID number of the job to be released.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
int32_t job_id = 12;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

papiServiceDestroy(handle);
```

See Also. papiJobHold

7.7. papiJobRestart

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

Syntax.

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

Inputs.

handle

Handle to the print service to use.

printer_name

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

job_id

The ID number of the job to be restarted.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
int32_t job_id = 12;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

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

papiServiceDestroy(handle);
```

See Also. papiPrinterListJobs

7.8. papiJobFree

Description. Free a job object.

Syntax.

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

Inputs.

job

Pointer to the printer object to free.

Outputs. none

Returns. none

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_job_t* job = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiJobQuery(handle,
                      printer_name,
                      12,
                      &job);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiJobQuery failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}

if (job != NULL)
{
    /* process the job object */
    ...
    papiJobFree(job);
}

papiServiceDestroy(handle);
```

See Also. papiJobQuery

7.9. papiJobListFree

Description. Free a list of job objects.

Syntax.

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

Inputs.

jobs

Pointer to the printer object list to free.

Outputs. none

Returns. none

Example.

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_printer_t** printers = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);
if (status != PAPI_OK)
{
    /* handle the error */
    ...
}

status = papiPrinterListJobs(handle,
                             printer_name,
                             NULL,
                             0, 0, 0,
                             &jobs);
if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiPrinterListJobs failed: %s\n",
            papiServiceGetStatusMessage(handle));
    ...
}

if (jobs != NULL)
{
    /* process the job objects */
    ...
    papiJobListFree(jobs);
}

papiServiceDestroy(handle);
```

See Also. papiPrinterListJobs

Chapter 8. Miscellaneous API

8.1. papiStatusString

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).

The returned message will be localized in the language of the submittor of the requestor.

Syntax.

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

Inputs.

status

The status value to convert to a status string.

Outputs. none

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

Example.

```
#include "papi.h"

papi_status_t status;
...
fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));
```

See Also. papiServiceGetStatusMessage

Chapter 9. Attributes

* *ISSUE: Waiting for reference to single document from Tom H.*

9.1. Extension Attributes

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

9.1.1. job-ticket-formats-supported

(1setOf type2 keyword) This optional printer atttribute 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.

* *ISSUE: I took the following required attr lists directly from IPP RFC 2911 to use as a starting point. We probably want to add/delete attrs from the lists.*

9.2. 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.

attributes-charset (?)
attributes-natural-language (?)
job-id
job-name
job-originating-user-name
job-printer-up-time
job-printer-uri
job-state
job-state-reasons
job-uri
time-at-creation
time-at-processing

time-at-completed

9.3. 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

Appendix A. Change History

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

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

Version 0.1 (April 3, 2002).

- Original draft version

End of Document