

1

2

Open Standard Print API (PAPI)

3

Version 0.3 (DRAFT)

4

5

Alan Hlava

6

IBM Printing Systems Division

7

Norm Jacobs

8

Sun Microsystems, Inc.

9

Michael R Sweet

10

Easy Software Products

11
12 **Open Standard Print API (PAPI): Version 0.3 (DRAFT)**

13 by Alan Hlava, Norm Jacobs, and Michael R Sweet

14 Version 0.3 (DRAFT) Edition

15 Copyright © 2002 by Free Standards Group

16 Permission to use, copy, modify and distribute this document for any purpose and without fee is hereby granted in
17 perpetuity, provided that the above copyright notice and this paragraph appear in all copies.

Table of Contents

19	1. Introduction.....	1
20	2. Print System Model.....	2
21	2.1. Introduction.....	2
22	2.2. Model.....	2
23	2.2.1. Print Service.....	3
24	2.2.2. Printer.....	3
25	2.2.3. Job.....	3
26	2.3. Security.....	3
27	2.3.1. Authentication.....	3
28	2.3.2. Authorization.....	4
29	2.3.3. Encryption.....	4
30	3. Common Structures.....	5
31	3.1. Conventions.....	5
32	3.2. Service Object (papi_service_t).....	5
33	3.3. Attributes and Values.....	5
34	3.4. Job Object (papi_job_t).....	7
35	3.5. Printer Object (papi_printer_t).....	7
36	3.6. Job Ticket (papi_job_ticket_t).....	7
37	3.7. Status (papi_status_t).....	8
38	3.8. List Filter (papi_filter_t).....	9
39	4. Service API.....	11
40	4.1. papiServiceCreate.....	11
41	4.2. papiServiceDestroy.....	13
42	4.3. papiServiceSetUsername.....	14
43	4.4. papiServiceSetPassword.....	15
44	4.5. papiServiceSetEncryption.....	17
45	4.6. papiServiceSetAuthCB.....	18
46	4.7. papiServiceSetAppData.....	19
47	4.8. papiServiceGetServicename.....	21
48	4.9. papiServiceGetUsername.....	21
49	4.10. papiServiceGetPassword.....	22
50	4.11. papiServiceGetEncryption.....	23
51	4.12. papiServiceGetAppData.....	24
52	4.13. papiServiceGetStatusMessage.....	25

53	5. Printer API	27
54	5.1. Usage.....	27
55	5.2. papiPrintersList.....	27
56	5.3. papiPrinterQuery.....	29
57	5.4. papiPrinterPause	31
58	5.5. papiPrinterResume.....	33
59	5.6. papiPrinterPurgeJobs	34
60	5.7. papiPrinterListJobs	35
61	5.8. papiPrinterFree.....	38
62	5.9. papiPrinterListFree	39
63	6. Attributes API.....	41
64	6.1. papiAttributeAdd	41
65	6.2. papiAttributeAddString.....	43
66	6.3. papiAttributeAddInteger	44
67	6.4. papiAttributeAddBoolean.....	46
68	6.5. papiAttributeAddRange	47
69	6.6. papiAttributeAddResolution.....	49
70	6.7. papiAttributeAddDatetime.....	50
71	6.8. papiAttributeListFree.....	52
72	6.9. papiAttributeListFind.....	52
73	6.10. papiAttributeListGetNext	53
74	7. Job API	55
75	7.1. papiJobSubmit.....	55
76	7.2. papiJobValidate.....	57
77	7.3. papiJobQuery	59
78	7.4. papiJobCancel	61
79	7.5. papiJobHold	62
80	7.6. papiJobRelease.....	64
81	7.7. papiJobRestart.....	65
82	7.8. papiJobFree	67
83	7.9. papiJobListFree.....	68
84	8. Miscellaneous API.....	70
85	8.1. papiStatusString	70
86	9. Attributes	71
87	9.1. Extension Attributes.....	71
88	9.1.1. job-ticket-formats-supported.....	71
89	9.2. Required Job Attributes	71
90	9.3. Required Printer Attributes	72

92 Chapter 1. Introduction

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

100 This version of the document describes stage 1 and stage 2 of the Open Standard Print
101 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.) |

102

103

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

106 Chapter 2. Print System Model

107 2.1. Introduction

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

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

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

118
119 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	

120

121

122 2.2. Model

123 The model on which the Open Standard Print API is derived from are the semantics
124 defined by the Internet Print Protocol (IPP) standard. This is a fairly simple model in
125 terms of the number of object types. It is defined very clearly and in detail in the IPP
126 RFC 2911, Chapter 2 (<http://ietf.org/rfc/rfc2911.txt?number=2911>).

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

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

132 **2.2.1. Print Service**

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

137 **2.2.2. Printer**

138 Printer objects are the target of print job requests. A printer object may represent an
139 actual printer (if the printer itself supports PAPI), an object in a server representing an
140 actual printer, or an abstract object in a server (perhaps representing a pool or class of
141 printers). Printer objects are identified via one or more names which may be short,
142 local names (such as "prtr1") or longer global names (such as a URI like
143 "http://printserv.mycompany.com:631/printers/prtr1"). The PAPI implementation may
144 detect and map short names to long global names in an implementation-specific way.

145 **2.2.3. Job**

146 Job objects are created after a successful print submission. They contain a set of
147 attributes describing the job and specifying how it will be printed, and they contain
148 (logically) the print data itself in the form of one or more "documents".

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

152 **2.3. Security**

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

155 **2.3.1. Authentication**

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

162 If the user name and password are not passed on the API call, the call may fail with an
163 error code indicating a security problem (e.g. PAPI_NOT_AUTHENTICATED).
164 See RFC 2616 and RFC 2617 for further details about HTTP security.

165 **2.3.2. Authorization**

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

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

173 **2.3.3. Encryption**

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

178 Chapter 3. Common Structures

179 3.1. Conventions

180

- 181 • All "char*" variables and fields are pointers to standard C/C++ NULL-terminated
182 strings.
- 183 • All pointer arrays (e.g. "char**") are assumed to be terminated by NULL pointers.
184 That is, the valid elements of the array are followed by an element containing a
185 NULL pointer that marks the end of the list.

186

187 3.2. Service Object (papi_service_t)

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

```
191 typedef void* papi_service_t;  
192
```

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

```
195 typedef enum  
196 {  
197     PAPI_ENCRYPT_IF_REQUESTED, /* Encrypt if requested (TLS upgrade) */  
198     PAPI_ENCRYPT_NEVER,      /* Never encrypt */  
199     PAPI_ENCRYPT_REQUIRED, /* Encryption is required (TLS upgrade) */  
200     PAPI_ENCRYPT_ALWAYS     /* Always encrypt (SSL) */  
201 } papi_encryption_t;  
202
```

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

205 3.3. Attributes and Values

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

Chapter 3. Common Structures

```
208     /* Attribute Type */
209     typedef enum
210     {
211         PAPI_STRING,
212         PAPI_INTEGER,
213         PAPI_BOOLEAN,
214         PAPI_RANGE,
215         PAPI_RESOLUTION,
216         PAPI_DATETIME
217     } papi_attribute_value_type_t;
218
```

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

```
220     /* Attribute Value */
221     typedef union
222     {
223         char* string;      /* PAPI_STRING value */
224
225         int integer;      /* PAPI_INTEGER value */
226
227         char boolean;     /* PAPI_BOOLEAN value */
228
229         struct            /* PAPI_RANGE value */
230         {
231             int lower;
232             int upper;
233         } range;
234
235         struct            /* PAPI_RESOLUTION value */
236         {
237             int xres;
238             int yres;
239         } resolution;
240
241         time_t datetime; /* PAPI_DATETIME value */
242     } papi_attribute_value_t;
243
```

```
244     /* Attribute and Values */
245     typedef struct
246     {
247         char* name;          /* attribute name */
248         papi_attribute_value_type_t type; /* type of values */
249         papi_attribute_value_t** values; /* list of values */
250     } papi_attribute_t;
251
```

```
252     /* Attribute update types */
253     #define PAPI_ATTR_APPEND 0x0001 /* Add values to attr */
254     #define PAPI_ATTR_REPLACE 0x0002 /* Delete existing
255                                     values then add new ones */
256     #define PAPI_ATTR_EXCL 0x0004 /* Fail if attr exists */
257
```

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

259 3.4. Job Object (`papi_job_t`)

260 This structure represents a job object.

```
261 typedef struct
262 {
263     char* name;
264     int32_t id;
265     papi_attribute_t** attributes;
266     papi_job_ticket_t* job_ticket;
267 } papi_job_t;
268
```

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

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

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

273 The "job_ticket" field points to a structure representing the job's associated job ticket.

274 A NULL value indicates there is no associated job ticket.

275 3.5. Printer Object (`papi_printer_t`)

276 This structure represents a printer object.

```
277 typedef struct
278 {
279     char* name;
280     papi_attribute_t** attributes;
281 } papi_printer_t;
282
```

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

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

285 3.6. Job Ticket (`papi_job_ticket_t`)

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

```
289 /* Job Ticket Format */
290 typedef enum
291 {
292     PAPI_JT_FORMAT_JDF = 0,          /* Job Definition Format */
293 } papi_jt_format_t;
```

294

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

```

296     /* Job Ticket */
297     typedef struct papi_job_ticket_s
298     {
299         papi_jt_format_t format, /* Format of job ticket */
300         char* ticket_data, /* Buffer containing the job
301                             ticket data. If NULL,
302                             uri must be specified */
303         char* uri, /* URI of the file containing
304                    the job ticket data. If
305                    ticket_data is specified, then
306                    uri is ignored. */
307     } papi_job_ticket_t;
308

```

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

310 3.7. Status (papi_status_t)

```

311     typedef enum
312     {
313         PAPI_OK = 0x0000,
314         PAPI_OK_SUBST,
315         PAPI_OK_CONFLICT,
316         PAPI_OK_IGNORED_SUBSCRIPTIONS,
317         PAPI_OK_IGNORED_NOTIFICATIONS,
318         PAPI_OK_TOO_MANY_EVENTS,
319         PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
320         PAPI_REDIRECTION_OTHER_SITE = 0x300,
321         PAPI_BAD_REQUEST = 0x0400,
322         PAPI_FORBIDDEN,
323         PAPI_NOT_AUTHENTICATED,
324         PAPI_NOT_AUTHORIZED,
325         PAPI_NOT_POSSIBLE,
326         PAPI_TIMEOUT,
327         PAPI_NOT_FOUND,
328         PAPI_GONE,
329         PAPI_REQUEST_ENTITY,
330         PAPI_REQUEST_VALUE,
331         PAPI_DOCUMENT_FORMAT,
332         PAPI_ATTRIBUTES,
333         PAPI_URI_SCHEME,
334         PAPI_CHARSET,
335         PAPI_CONFLICT,
336         PAPI_COMPRESSION_NOT_SUPPORTED,
337         PAPI_COMPRESSION_ERROR,
338         PAPI_DOCUMENT_FORMAT_ERROR,
339         PAPI_DOCUMENT_ACCESS_ERROR,
340         PAPI_ATTRIBUTES_NOT_SETTABLE,
341         PAPI_IGNORED_ALL_SUBSCRIPTIONS,
342         PAPI_TOO_MANY_SUBSCRIPTIONS,
343         PAPI_IGNORED_ALL_NOTIFICATIONS,
344         PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
345         PAPI_INTERNAL_ERROR = 0x0500,
346         PAPI_OPERATION_NOT_SUPPORTED,
347         PAPI_SERVICE_UNAVAILABLE,

```

```

348     PAPI_VERSION_NOT_SUPPORTED,
349     PAPI_DEVICE_ERROR,
350     PAPI_TEMPORARY_ERROR,
351     PAPI_NOT_ACCEPTING,
352     PAPI_PRINTER_BUSY,
353     PAPI_ERROR_JOB_CANCELLED,
354     PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
355     PAPI_PRINTER_IS_DEACTIVATED,
356     PAPI_BAD_ARGUMENT
357 } papi_status_t;
358

```

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

361 3.8. List Filter (papi_filter_t)

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

```

365 typedef enum
366 {
367     PAPI_FILTER_BITMASK = 0
368     /* future filter types may be added here */
369 } papi_filter_type_t;
370
371 typedef struct
372 {
373     papi_filter_type_t    type; /* Type of filter specified */
374
375     union
376     {
377         unsigned int    mask; /* PAPI_FILTER_BITMASK */
378
379         /* future filter types may be added here */
380     } u;
381 } papi_filter_t;
382

```

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

```

385 enum
386 {
387     PAPI_PRINTER_LOCAL = 0x0000, /* Local printer or class */
388     PAPI_PRINTER_CLASS = 0x0001, /* Printer class */
389     PAPI_PRINTER_REMOTE = 0x0002, /* Remote printer or class */
390     PAPI_PRINTER_BW = 0x0004, /* Can do B&W printing */
391     PAPI_PRINTER_COLOR = 0x0008, /* Can do color printing */
392     PAPI_PRINTER_DUPLEX = 0x0010, /* Can do duplexing */
393     PAPI_PRINTER_STAPLE = 0x0020, /* Can staple output */
394     PAPI_PRINTER_COPIES = 0x0040, /* Can do copies */
395     PAPI_PRINTER_COLLATE = 0x0080, /* Can collage copies */
396     PAPI_PRINTER_PUNCH = 0x0100, /* Can punch output */
397     PAPI_PRINTER_COVER = 0x0200, /* Can cover output */

```

Chapter 3. Common Structures

```
398         PAPI_PRINTER_BIND = 0x0400,          /* Can bind output */
399         PAPI_PRINTER_SORT = 0x0800,          /* Can sort output */
400         PAPI_PRINTER_SMALL = 0x1000,        /* Can do Letter/Legal/A4 */
401         PAPI_PRINTER_MEDIUM = 0x2000,      /* Can do Tabloid/B/C/A3/A2 */
402         PAPI_PRINTER_LARGE = 0x4000,       /* Can do D/E/A1/A0 */
403         PAPI_PRINTER_VARIABLE = 0x8000,    /* Can do variable sizes */
404         PAPI_PRINTER_IMPLICIT = 0x10000,   /* Implicit class */
405         PAPI_PRINTER_DEFAULT = 0x20000,    /* Default printer on network */
406         PAPI_PRINTER_OPTIONS = 0xfffc     /* ~(CLASS | REMOTE | IMPLICIT) */
407     };
408
```

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

410 Chapter 4. Service API

411 4.1. papiServiceCreate

412 **Description.** Create a print service handle to be used in subsequent calls. Memory is
413 allocated and copies of the input arguments are created so that the handle can be used
414 outside the scope of the input variables. The caller must call papiServiceDestroy when
415 done in order to free the resources associated with the print service handle.

416 **Syntax.**

```
417 papi_status_t papiServiceCreate(  
418     papi_service_t*      handle,  
419     const char*          service_name,  
420     const char*          user_name,  
421     const char*          password,  
422     int (*authCB) (papi_service_t svc),  
423     const papi_encryption_t encryption,  
424     void*                 app_data );  
425
```

426

427 **Inputs.**

428 service_name

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

433 user_name

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

437 password

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

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

446 encryption

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

448 app_data

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

451

452 **Outputs.**

453 handle

454 A print service handle to be used on subsequent API calls. The handle will
 455 always be set to something even if the function fails, in which case it may be set
 456 to NULL.

457

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

460 **Example.**

```

461           #include "papi.h"
462
463           papi_status_t status;
464           papi_service_t handle = NULL;
465           const char* service_name = "ipp:/printserv:631";
466           const char* user_name = "pappy";
467           const char* password = "goober";
468           ...
469           status = papiServiceCreate(&handle,
470                                      service_name,
471                                      user_name,
472                                      password,
473                                      NULL,
474                                      PAPI_ENCRYPT_IF_REQUESTED,
475                                      NULL);
476           if (status != PAPI_OK)
477           {
478                 /* handle the error */
479                 fprintf(stderr, "papiServiceCreate failed: %s\n",
```

```

480         papiStatusString(status));
481     if (handle != NULL)
482     {
483         fprintf(stderr, "    details: %s\n",
484             papiServiceGetStatusMessage(handle));
485     }
486     ...
487 }
488 ...
489 papiServiceDestroy(handle);
490

```

491

492 **See Also.** `papiServiceDestroy`, `papiServiceGetStatusMessage`,
493 `papiServiceSetUsername`, `papiServiceSetPassword`, `papiServiceSetEncryption`,
494 `papiServiceSetAuthCB`

495 4.2. `papiServiceDestroy`

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

499 **Syntax.**

```

500 void papiServiceDestroy(
501     papi_service_t handle );
502

```

503

504 **Inputs.**

505 handle

506 The print service handle to be destroyed.

507

508 **Outputs.** none

509 **Returns.** none

510 **Example.**

```

511 #include "papi.h"
512
513 papi_status_t status;
514 papi_service_t handle = NULL;
515 const char* service_name = "ipp://printserv:631";
516 const char* user_name = "pappy";

```

```
517     const char* password = "goober";
518     ...
519     status = papiServiceCreate(&handle,
520                               service_name,
521                               user_name,
522                               password,
523                               NULL,
524                               PAPI_ENCRYPT_IF_REQUESTED,
525                               NULL);
526
527     if (status != PAPI_OK)
528     {
529         /* handle the error */
530         ...
531     }
532     papiServiceDestroy(handle);
533
```

534

535 **See Also.** papiServiceCreate

536 4.3. papiServiceSetUsername

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

540 **Syntax.**

```
541     papi_status_t papiServiceSetUsername(  
542         papi_service_t handle,  
543         const char* user_name );  
544
```

545

546 **Inputs.**

547 handle

548 Handle to the print service to update.

549 user_name

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

553

554 **Outputs.** handle is updated.

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

557 **Example.**

```
558 #include "papi.h"
559
560 papi_status_t status;
561 papi_service_t handle = NULL;
562 const char* user_name = "pappy";
563 ...
564 status = papiServiceCreate(&handle,
565                             NULL,
566                             NULL,
567                             NULL,
568                             NULL,
569                             PAPI_ENCRYPT_IF_REQUESTED,
570                             NULL);
571
572 if (status != PAPI_OK)
573 {
574     /* handle the error */
575     ...
576 }
577
578 status = papiServiceSetUsername(handle, user_name);
579 if (status != PAPI_OK)
580 {
581     /* handle the error */
582     fprintf(stderr, "papiServiceSetUsername failed: %s\n",
583             papiServiceGetStatusMessage(handle));
584     ...
585 }
586 papiServiceDestroy(handle);
587
```

588

589 **See Also.** papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

590 4.4. papiServiceSetPassword

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

594 **Syntax.**

```
595 papi_status_t papiServiceSetPassword(
596     papi_service_t handle,
597     const char* password );
598
```

599

600 **Inputs.**

601 handle

602 Handle to the print service to update.

603 password

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

605

606 **Outputs.** handle is updated.607 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
608 failure value is returned.609 **Example.**

```

610       #include "papi.h"
611
612       papi_status_t status;
613       papi_service_t handle = NULL;
614       const char* password = "goober";
615       ...
616       status = papiServiceCreate(&handle,
617                                  NULL,
618                                  NULL,
619                                  NULL,
620                                  NULL,
621                                  PAPI_ENCRYPT_IF_REQUESTED,
622                                  NULL);
623
624       if (status != PAPI_OK)
625       {
626             /* handle the error */
627             ...
628       }
629
630       status = papiServiceSetPassword(handle, password);
631       if (status != PAPI_OK)
632       {
633             /* handle the error */
634             fprintf(stderr, "papiServiceSetPassword failed: %s\n",
635                      papiServiceGetStatusMessage(handle));
636             ...
637       }
638       ...
639       papiServiceDestroy(handle);

```

640

641 **See Also.** papiServiceCreate, papiServiceSetUsername, papiServiceGetStatusMessage

642 4.5. papiServiceSetEncryption

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

645 **Syntax.**

```
646 papi_status_t papiServiceSetEncryption(
647     papi_service_t handle,
648     const papi_encryption_t encryption );
649
```

650

651 **Inputs.**

652 handle

653 Handle to the print service to update.

654 encryption

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

656

657 **Outputs.** handle is updated.

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

660 **Example.**

```
661 #include "papi.h"
662
663 papi_status_t status;
664 papi_service_t handle = NULL;
665 ...
666 status = papiServiceCreate(&handle,
667     NULL,
668     NULL,
669     NULL,
670     NULL,
671     PAPI_ENCRYPT_IF_REQUESTED,
672     NULL);
673
674 if (status != PAPI_OK)
675 {
676     /* handle the error */
677     ...
678 }
679
680 status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
681 if (status != PAPI_OK)
```

```
681     {
682         /* handle the error */
683         fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
684                papiServiceGetStatusMessage(handle));
685         ...
686     }
687     ...
688     papiServiceDestroy(handle);
689
```

690

691 **See Also.** `papiServiceCreate`, `papiServiceGetStatusMessage`

692 4.6. `papiServiceSetAuthCB`

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

695 **Syntax.**

```
696 papi_status_t papiServiceSetAuthCB(
697     papi_service_t handle,
698     const int (*authCB)(papi_service_t svc) );
699
```

700

701 **Inputs.**

702 `handle`

703 Handle to the print service to update.

704 `authCB`

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

710

711 **Outputs.** `handle` is updated.

712 **Returns.** If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate
713 failure value is returned.

714 **Example.**


```

715     #include "papi.h"
716
717     extern int get_password(papi_service_t handle);
718     papi_status_t status;
719     papi_service_t handle = NULL;
720     ...
721     status = papiServiceCreate(&handle,
722                               NULL,
723                               NULL,
724                               NULL,
725                               NULL,
726                               PAPI_ENCRYPT_IF_REQUESTED,
727                               NULL);
728
729     if (status != PAPI_OK)
730     {
731         /* handle the error */
732         ...
733     }
734
735     status = papiServiceSetAuthCB(handle, get_password);
736     if (status != PAPI_OK)
737     {
738         /* handle the error */
739         fprintf(stderr, "papiServiceSetAuthCB failed: %s\n",
740                papiServiceGetStatusMessage(handle));
741         ...
742     }
743     ...
744     papiServiceDestroy(handle);

```

745

746 **See Also.** `papiServiceCreate`, `papiServiceGetStatusMessage`

747 4.7. `papiServiceSetAppData`

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

751 **Syntax.**

```

752     papi_status_t papiServiceSetAppData(
753         papi_service_t handle,
754         const void*     app_data );
755

```

756

757 **Inputs.**

758 handle

759 Handle to the print service to update.

760 app_data

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

763

764 **Outputs.** handle is updated.

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

767 **Example.**

```

768 #include "papi.h"
769
770 extern int get_password(papi_service_t handle);
771 papi_status_t status;
772 papi_service_t handle = NULL;
773 char* app_data = "some data";
774 ...
775 status = papiServiceCreate(&handle,
776                             NULL,
777                             NULL,
778                             NULL,
779                             NULL,
780                             PAPI_ENCRYPT_IF_REQUESTED,
781                             NULL);
782 if (status != PAPI_OK)
783 {
784     /* handle the error */
785     ...
786 }
787
788 status = papiServiceSetAppData(handle, app_data);
789 if (status != PAPI_OK)
790 {
791     /* handle the error */
792     fprintf(stderr, "papiServiceSetAppData failed: %s\n",
793            papiServiceGetStatusMessage(handle));
794     ...
795 }
796 ...
797 papiServiceDestroy(handle);
798

```

799

800 **See Also.** papiServiceCreate, papiServiceGetStatusMessage

801 4.8. papiServiceGetServicename

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

803 **Syntax.**

```
804 char* papiServiceGetServicename (
805     papi_service_t handle );
806
```

807

808 **Inputs.**

809 handle

810 Handle to the print service.

811

812 **Outputs.** none

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

814 **Example.**

```
815 #include "papi.h"
816
817 papi_status_t status;
818 papi_service_t handle = NULL;
819 char* service_name = NULL;
820 ...
821 service_name = papiServiceGetServicename(handle);
822 if (service_name != NULL)
823 {
824     /* use the returned name */
825     ...
826 }
827 ...
828 papiServiceDestroy(handle);
829
```

830

831 **See Also.** papiServiceCreate

832 4.9. papiServiceGetUsername

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

834 **Syntax.**

```
835     char* papiServiceGetUsername (  
836         papi_service_t handle );  
837
```

838

839 **Inputs.**

840 handle

841 Handle to the print service.

842

843 **Outputs.** none

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

845 **Example.**

```
846     #include "papi.h"  
847  
848     papi_status_t status;  
849     papi_service_t handle = NULL;  
850     char* user_name = NULL;  
851     ...  
852     user_name = papiServiceGetUsername(handle);  
853     if (user_name != NULL)  
854     {  
855         /* use the returned name */  
856         ...  
857     }  
858     ...  
859     papiServiceDestroy(handle);  
860
```

861

862 **See Also.** papiServiceCreate, papiServiceSetUsername

863 4.10. papiServiceGetPassword

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

865 **Syntax.**

```
866     char* papiServiceGetPassword(  
867         papi_service_t handle );  
868
```

869

870 **Inputs.**

871 handle

872 Handle to the print service.

873

874 **Outputs.** none

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

876 **Example.**

```

877       #include "papi.h"
878
879       papi_status_t status;
880       papi_service_t handle = NULL;
881       char* password = NULL;
882       ...
883       password = papiServiceGetPassword(handle);
884       if (password != NULL)
885       {
886           /* use the returned password */
887           ...
888       }
889       ...
890       papiServiceDestroy(handle);
891
```

892

893 **See Also.** papiServiceCreate, papiServiceSetPassword

894 4.11. papiServiceGetEncryption

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

896 **Syntax.**

```

897       papi_encryption_t papiServiceGetEncryption(
898           papi_service_t handle );
899
```

900

901 **Inputs.**

902 handle

903 Handle to the print service.

904

905 **Outputs.** none

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

907 **Example.**

```
908           #include "papi.h"
909
910           papi_status_t status;
911           papi_service_t handle = NULL;
912           papi_encryption_t encryption;
913           ...
914           encryption = papiServiceGetEncryption(handle);
915           /* use the returned encryption value */
916           ...
917           papiServiceDestroy(handle);
918
```

919

920 **See Also.** papiServiceCreate, papiServiceSetEncryption

921 **4.12. papiServiceGetAppData**

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

924 **Syntax.**

```
925           void* papiServiceGetAppData(
926                     papi_service_t handle );
927
```

928

929 **Inputs.**

930 handle

931 Handle to the print service.

932

933 **Outputs.** none

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

936 **Example.**

```
937           #include "papi.h"
938
```

```

939     papi_status_t status;
940     papi_service_t handle = NULL;
941     char* app_data = NULL;
942     ...
943     app_data = (char*)papiServiceGetAppData(handle);
944     if (app_data != NULL)
945     {
946         /* use the returned application data */
947         ...
948     }
949     ...
950     papiServiceDestroy(handle);
951

```

952

953 **See Also.** `papiServiceCreate`, `papiServiceSetAppData`

954 4.13. `papiServiceGetStatusMessage`

955 **Description.** Get the message associated with the status of the last operation
 956 performed. The status message returned from this function may be more detailed than
 957 the status message returned from `papiStatusString` (if the print service supports
 958 returning more detailed error messages).

959 The returned message will be localized in the language of the submitter of the original
 960 operation.

961 **Syntax.**

```

962     const char* papiServiceGetStatusMessage(
963         const papi_service_t handle );
964

```

965

966 **Inputs.**967 `handle`

968 Handle to the print service.

969

970 **Outputs.** none971 **Returns.** Pointer to the message associated with the status of the last operation
 972 performed.973 **Example.**

```
974     #include "papi.h"
975
976     papi_status_t status;
977     papi_service_t handle = NULL;
978     const char* user_name = "pappy";
979     ...
980     status = papiServiceCreate(&handle,
981                               NULL,
982                               NULL,
983                               NULL,
984                               NULL,
985                               PAPI_ENCRYPT_IF_REQUESTED,
986                               NULL);
987
988     if (status != PAPI_OK)
989     {
990         /* handle the error */
991         ...
992     }
993
994     status = papiServiceSetUsername(handle, user_name);
995     if (status != PAPI_OK)
996     {
997         /* handle the error */
998         fprintf(stderr, "papiServiceSetUsername failed: %s\n",
999                papiServiceGetStatusMessage(handle));
1000        ...
1001    }
1002    ...
1003    papiServiceDestroy(handle);
```

1004

1005 **See Also.** `papiStatusString`

1006 Chapter 5. Printer API

1007 5.1. Usage

1008 The papiPrinterQuery function queries all/some of the attributes of a printer object. It
1009 returns a list of printer attributes. A successful call to papiPrinterQuery is typically
1010 followed by code which examines and processes the returned attributes. The using
1011 program would then call papiPrinterFree to delete the returned results.

1012 Printers can be found via calls to papiPrintersList. A successful call to papiPrintersList
1013 is typically followed by code to iterate through the list of returned printers, possibly
1014 querying each (papiPrinterQuery) for further information (e.g. to restrict what printers
1015 get displayed for a particular user/request). The using program would then call
1016 papiPrinterListFree to free the returned results.

1017 5.2. papiPrintersList

1018 **Description.** List all printers known by the print service which match the specified
1019 filter.

1020 Depending on the functionality of the target service's "printer directory", the returned
1021 list may be limited to only printers managed by a particular server or it may include
1022 printers managed by other servers.

1023 **Syntax.**

```
1024 papi_status_t papiPrintersList(  
1025             papi_service_t    handle,  
1026             const char*       requested_attrs[],  
1027             const papi_filter_t* filter,  
1028             papi_printer_t*** printers );  
1029
```

1030

1031 **Inputs.**

1032 handle

1033 Handle to the print service to use.

1034 requested_attrs
 1035 (optional) NULL terminated array of attribute names to be queried. If NULL is
 1036 passed then all available attributes should be returned.

1037 filter
 1038 (optional) Pointer to a filter to limit the number of printers returned on the list
 1039 request. See Section 3.8 for details. If NULL is passed then all known printers are
 1040 listed.

1041

1042 Outputs.

1043 printers
 1044 List of printer objects that matched the filter criteria.

1045

1046 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 1047 failure value is returned.

1048 Example.

```

1049 #include "papi.h"
1050
1051 int i;
1052 papi_status_t status;
1053 papi_service_t handle = NULL;
1054 const char* service_name = "ipp://printserv:631";
1055 const char* user_name = "pappy";
1056 const char* password = "goober";
1057 const char* req_attrs[] =
1058 {
1059     "printer-name",
1060     "printer-location",
1061     NULL
1062 };
1063 const papi_filter_t filter =
1064     PAPI_PRINTER_BW | PAPI_PRINTER_DUPLEX;
1065 papi_printer_t** printers = NULL;
1066 ...
1067 status = papiServiceCreate(&handle,
1068                             service_name,
1069                             user_name,
1070                             password,
1071                             NULL,
1072                             PAPI_ENCRYPT_IF_REQUESTED,
1073                             NULL);
1074 if (status != PAPI_OK)
1075 {
1076     /* handle the error */
1077     ...
1078 }
1079

```

```

1080     status = papiPrinterList(handle,
1081                             req_attrs,
1082                             filter,
1083                             &printers);
1084     if (status != PAPI_OK)
1085     {
1086         /* handle the error */
1087         fprintf(stderr, "papiPrinterList failed: %s\n",
1088                 papiServiceGetStatusMessage(handle));
1089         ...
1090     }
1091
1092     if (printers != NULL)
1093     {
1094         for (i=0; printers[i] != NULL; i++)
1095         {
1096             /* process the printer object */
1097             ...
1098         }
1099         papiPrinterListFree(printers);
1100     }
1101
1102     papiServiceDestroy(handle);
1103

```

1104

1105 **See Also.** `papiPrinterListFree`, `papiPrinterQuery`

1106 5.3. `papiPrinterQuery`

1107 **Description.** Queries some or all the attributes of the specified printer object. This
1108 includes attributes representing the capabilities of the printer, which the caller may use
1109 to determine which print options to present to the user. How the attributes are obtained
1110 (e.g. from a static database, from a dialog with the hardware, from a dialog with a
1111 driver, etc.) is up to the implementer of the API and is beyond the scope of this
1112 standard.

1113 **Syntax.**

```

1114     papi_status_t papiPrinterQuery(
1115                 papi_service_t   handle,
1116                 const char*       name,
1117                 const char*       requested_attrs[],
1118                 papi_printer_t**  printer );
1119

```

1120

1121 **Inputs.**

1122 handle

1123 Handle to the print service to use.

1124 name

1125 The name or URI of the printer to query.

1126 requested_attrs

1127 (optional) NULL terminated array of attributes to be queried. If NULL is passed

1128 then all attributes are queried. (NOTE: The printer may return more attributes

1129 than you requested. This is merely an advisory request that may reduce the

1130 amount of data returned if the printer/server supports it.)

1131

1132 **Outputs.**

1133 printer

1134 Pointer to a printer object containing the requested attributes.

1135

1136 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate

1137 failure value is returned.

1138 **Example.**

```

1139           #include "papi.h"
1140
1141           papi_status_t status;
1142           papi_service_t handle = NULL;
1143           const char* service_name = "ipp://printserv:631";
1144           const char* user_name = "pappy";
1145           const char* password = "goober";
1146           const char* printer_name = "my-printer";
1147           const char* req_attrs[] =
1148           {
1149                 "printer-name",
1150                 "printer-location",
1151                 "printer-state",
1152                 "printer-state-reasons",
1153                 "printer-state-message",
1154                 NULL
1155           };
1156           papi_printer_t* printer = NULL;
1157           ...
1158           status = papiServiceCreate(&handle,
1159                                         service_name,
1160                                         user_name,
1161                                         password,
1162                                         NULL,
1163                                         PAPI_ENCRYPT_IF_REQUESTED,
1164                                         NULL);

```

```

1165     if (status != PAPI_OK)
1166     {
1167         /* handle the error */
1168         ...
1169     }
1170
1171     status = papiPrinterQuery(handle,
1172                             printer_name,
1173                             req_attrs,
1174                             &printer);
1175
1176     if (status != PAPI_OK)
1177     {
1178         /* handle the error */
1179         fprintf(stderr, "papiPrinterQuery failed: %s\n",
1180                papiServiceGetStatusMessage(handle));
1181         ...
1182     }
1183
1184     if (printer != NULL)
1185     {
1186         /* process the printer object */
1187         ...
1188         papiPrinterFree(printer);
1189     }
1190
1191     papiServiceDestroy(handle);

```

1192

1193 **See Also.** `papiPrinterList`, `papiPrinterFree`

1194 5.4. `papiPrinterPause`

1195 **Description.** Stops the printer object from scheduling jobs to be printed. Depending on
1196 the implementation, this operation may also stop the printer from processing the
1197 current job(s). This operation is optional and may not be supported by all
1198 printers/servers. Use `papiPrinterResume` to undo the effects of this operation.

1199 Depending on the implementation, this function may also stop the print service from
1200 processing currently printing job(s).

1201 **Syntax.**

```

1202     papi_status_t papiPrinterPause(
1203                 papi_service_t   handle,
1204                 const char*       name );
1205

```

1206

1207 **Inputs.**

1208 handle

1209 Handle to the print service to use.

1210 name

1211 The name or URI of the printer to operate on.

1212

1213 **Outputs.** none

1214 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate

1215 failure value is returned.

1216 **Example.**

```

1217 #include "papi.h"
1218
1219 papi_status_t status;
1220 papi_service_t handle = NULL;
1221 const char* service_name = "ipp://printserv:631";
1222 const char* user_name = "pappy";
1223 const char* password = "goober";
1224 const char* printer_name = "my-printer";
1225 ...
1226 status = papiServiceCreate(&handle,
1227                             service_name,
1228                             user_name,
1229                             password,
1230                             NULL,
1231                             PAPI_ENCRYPT_IF_REQUESTED,
1232                             NULL);
1233
1234 if (status != PAPI_OK)
1235 {
1236     /* handle the error */
1237     ...
1238 }
1239
1240 status = papiPrinterPause(handle, printer_name);
1241 if (status != PAPI_OK)
1242 {
1243     /* handle the error */
1244     fprintf(stderr, "papiPrinterPause failed: %s\n",
1245             papiServiceGetStatusMessage(handle));
1246     ...
1247 }
1248 ...
1249 papiServiceDestroy(handle);

```

1250

1251 **See Also.** papiPrinterResume

1252 5.5. papiPrinterResume

1253 **Description.** Requests that the printer resume scheduling jobs to be printed (i.e. it
 1254 undoes the effects of papiPrinterPause). This operation is optional and may not be
 1255 supported by all printers/servers, but it must be supported if papiPrinterPause is
 1256 supported.

1257 **Syntax.**

```
1258 papi_status_t papiPrinterResume (
1259             papi_service_t      handle,
1260             const char*         name );
1261
```

1262

1263 **Inputs.**

1264 handle

Handle to the print service to use.

1266 name

The name or URI of the printer to operate on.

1268

1269 **Outputs.** none

1270 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 1271 failure value is returned.

1272 **Example.**

```
1273 #include "papi.h"
1274
1275 papi_status_t status;
1276 papi_service_t handle = NULL;
1277 const char* service_name = "ipp://printserv:631";
1278 const char* user_name = "pappy";
1279 const char* password = "goober";
1280 const char* printer_name = "my-printer";
1281 ...
1282 status = papiServiceCreate(&handle,
1283                          service_name,
1284                          user_name,
1285                          password,
1286                          NULL,
1287                          PAPI_ENCRYPT_IF_REQUESTED,
1288                          NULL);
1289
1290 if (status != PAPI_OK)
```

```

1290     {
1291         /* handle the error */
1292         ...
1293     }
1294
1295     status = papiPrinterPause(handle, printer_name);
1296     if (status != PAPI_OK)
1297     {
1298         /* handle the error */
1299         fprintf(stderr, "papiPrinterPause failed: %s\n",
1300                papiServiceGetStatusMessage(handle));
1301         ...
1302     }
1303     ...
1304     status = papiPrinterResume(handle, printer_name);
1305     if (status != PAPI_OK)
1306     {
1307         /* handle the error */
1308         fprintf(stderr, "papiPrinterResume failed: %s\n",
1309                papiServiceGetStatusMessage(handle));
1310         ...
1311     }
1312
1313     papiServiceDestroy(handle);
1314

```

1315

1316 **See Also.** `papiPrinterPause`

1317 5.6. `papiPrinterPurgeJobs`

1318 **Description.** Remove all jobs from the specified printer object regardless of their
1319 states. This includes removing jobs that have completed and are being kept for history
1320 (if any). This operation is optional and may not be supported by all printers/servers.

1321 **Syntax.**

```

1322 papi_status_t papiPrinterPurgeJobs(
1323             papi_service_t    handle,
1324             const char*       name );
1325

```

1326

1327 **Inputs.**

1328 `handle`

1329 Handle to the print service to use.

1330 name
 1331 The name or URI of the printer to operate on.

1332

1333 **Outputs.** none

1334 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 1335 failure value is returned.

1336 **Example.**

```

1337 #include "papi.h"
1338
1339 papi_status_t status;
1340 papi_service_t handle = NULL;
1341 const char* service_name = "ipp://printserv:631";
1342 const char* user_name = "pappy";
1343 const char* password = "goober";
1344 const char* printer_name = "my-printer";
1345 ...
1346 status = papiServiceCreate(&handle,
1347                             service_name,
1348                             user_name,
1349                             password,
1350                             NULL,
1351                             PAPI_ENCRYPT_IF_REQUESTED,
1352                             NULL);
1353
1354 if (status != PAPI_OK)
1355 {
1356     /* handle the error */
1357     ...
1358 }
1359
1360 status = papiPrinterPurgeJobs(handle, printer_name);
1361 if (status != PAPI_OK)
1362 {
1363     /* handle the error */
1364     fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
1365             papiServiceGetStatusMessage(handle));
1366     ...
1367 }
1368
1369 papiServiceDestroy(handle);

```

1370

1371 **See Also.** papiJobCancel

1372 5.7. papiPrinterListJobs

1373 **Description.** List print job(s) associated with the specified printer.

1374 **Syntax.**

```

1375     papi_status_t papiPrinterListJobs(
1376         papi_service_t handle,
1377         const char* printer,
1378         const char* requested_attrs[],
1379         const int type_mask,
1380         const int max_num_jobs,
1381         papi_job_t*** jobs );
1382

```

1383

1384 **Inputs.**

1385 handle

1386 Handle to the print service to use.

1387 requested_attrs

1388 (optional) NULL terminated array of attributes to be queried. If NULL is passed
 1389 then all available attributes are queried. (NOTE: The printer may return more
 1390 attributes than you requested. This is merely an advisory request that may reduce
 1391 the amount of data returned if the printer/server supports it.)

1392 type_mask

1393 A bit mask which determines what jobs will get returned. The following
 1394 constants can be bitwise-OR-ed together to select which types of jobs to list:

```

1395 #define PAPI_LIST_JOBS_OTHERS      0x0001 /* return jobs other than
1396                                     those submitted by the
1397                                     user name assoc with
1398                                     the handle */
1399 #define PAPI_LIST_JOBS_COMPLETED  0x0002 /* return completed jobs */
1400 #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
1401                                     jobs */
1402 #define PAPI_LIST_JOBS_ALL        0xFFFF /* return all jobs */
1403

```

1404

1405 max_num_jobs

1406 Limit to the number of jobs returned. If 0 is passed, then there is no limit on the
 1407 number of jobs which may be returned.

1408

1409 **Outputs.**

1410 jobs

1411 List of job objects returned.

1412

1413 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
1414 failure value is returned.

1415 **Example.**

```

1416 #include "papi.h"
1417
1418 int i;
1419 papi_status_t status;
1420 papi_service_t handle = NULL;
1421 const char* printer_name = "my-printer";
1422 papi_printer_t** printers = NULL;
1423 const char* job_attrs[] =
1424 {
1425     "job-id",
1426     "job-name",
1427     "job-originating-user-name",
1428     "job-state",
1429     "job-state-reasons",
1430     NULL
1431 };
1432 ...
1433 status = papiServiceCreate(&handle,
1434                             NULL,
1435                             NULL,
1436                             NULL,
1437                             NULL,
1438                             PAPI_ENCRYPT_NEVER,
1439                             NULL);
1440
1441 if (status != PAPI_OK)
1442 {
1443     /* handle the error */
1444     ...
1445 }
1446
1447 status = papiPrinterListJobs(handle,
1448                               printer_name,
1449                               job_attrs,
1450                               PAPI_LIST_JOBS_ALL,
1451                               0,
1452                               &jobs);
1453
1454 if (status != PAPI_OK)
1455 {
1456     /* handle the error */
1457     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
1458             papiServiceGetStatusMessage(handle));
1459     ...
1460 }
1461
1462 if (jobs != NULL)
1463 {
1464     for(i=0; jobs[i] != NULL; i++)
1465     {
1466         /* process the job */
1467         ...

```

```

1466     }
1467     papiJobListFree(jobs);
1468 }
1469
1470 papiServiceDestroy(handle);
1471

```

1472

1473 **See Also.** papiJobQuery, papiJobListFree

1474 5.8. papiPrinterFree

1475 **Description.** Free a printer object.

1476 **Syntax.**

```

1477 void papiPrinterFree(
1478     papi_printer_t* printer );
1479

```

1480

1481 **Inputs.**

1482 printer

Pointer to the printer object to free.

1484

1485 **Outputs.** none

1486 **Returns.** none

1487 **Example.**

```

1488 #include "papi.h"
1489
1490 papi_status_t status;
1491 papi_service_t handle = NULL;
1492 const char* printer_name = "my-printer";
1493 papi_printer_t* printer = NULL;
1494 ...
1495 status = papiServiceCreate(&handle,
1496     NULL,
1497     NULL,
1498     NULL,
1499     NULL,
1500     PAPI_ENCRYPT_NEVER,
1501     NULL);
1502 if (status != PAPI_OK)
1503 {
1504     /* handle the error */

```

```

1505     ...
1506     }
1507
1508     status = papiPrinterQuery(handle,
1509                             printer_name,
1510                             NULL,
1511                             &printer);
1512
1513     if (status != PAPI_OK)
1514     {
1515         /* handle the error */
1516         fprintf(stderr, "papiPrinterQuery failed: %s\n",
1517                papiServiceGetStatusMessage(handle));
1518         ...
1519     }
1520
1521     if (printer != NULL)
1522     {
1523         /* process the printer object */
1524         ...
1525         papiPrinterFree(printer);
1526     }
1527
1528     papiServiceDestroy(handle);

```

1529

1530 **See Also.** `papiPrinterQuery`

1531 5.9. `papiPrinterListFree`

1532 **Description.** Free a list of printer objects.

1533 **Syntax.**

```

1534 void papiPrinterListFree(
1535     papi_printer_t**    printers );
1536

```

1537

1538 **Inputs.**

1539 `printers`

1540 Pointer to the printer object list to free.

1541

1542 **Outputs.** none

1543 **Returns.** none

1544 **Example.**

```
1545     #include "papi.h"
1546
1547     papi_status_t status;
1548     papi_service_t handle = NULL;
1549     const char* printer_name = "my-printer";
1550     papi_printer_t** printers = NULL;
1551     ...
1552     status = papiServiceCreate(&handle,
1553                               NULL,
1554                               NULL,
1555                               NULL,
1556                               NULL,
1557                               PAPI_ENCRYPT_NEVER,
1558                               NULL);
1559     if (status != PAPI_OK)
1560     {
1561         /* handle the error */
1562         ...
1563     }
1564
1565     status = papiPrinterList(handle,
1566                             NULL,
1567                             NULL,
1568                             &printers);
1569     if (status != PAPI_OK)
1570     {
1571         /* handle the error */
1572         fprintf(stderr, "papiPrinterList failed: %s\n",
1573                papiServiceGetStatusMessage(handle));
1574         ...
1575     }
1576
1577     if (printers != NULL)
1578     {
1579         /* process the printer objects */
1580         ...
1581         papiPrinterListFree(printers);
1582     }
1583
1584     papiServiceDestroy(handle);
1585
```

1586

1587 **See Also.** `papiPrinterList`

1588 Chapter 6. Attributes API

1589 6.1. papiAttributeAdd

1590 **Description.** Add an attribute/value to an attribute list. Memory is allocated and copies
1591 of the input arguments are created. It is the caller's responsibility to call
1592 papiAttributeListFree when done with the attribute list.

1593 This function is equivalent to the papiAttributeAddString, papiAttributeAddInteger,
1594 etc. functions defined later in this chapter.

1595 **Syntax.**

```
1596 papi_status_t papiAttributeAdd(  
1597     papi_attribute_t*** attrs,  
1598     const char* name,  
1599     const papi_attribute_value_type_t type,  
1600     const int update_type,  
1601     const int num_values,  
1602     ... );  
1603
```

1604

1605 **Inputs.**

1606 attrs

1607 Points to an attribute list. If a NULL value is passed, this function will allocate
1608 the attribute list.

1609 name

1610 Points to the name of the attribute to add.

1611 type

1612 The type of values for this attribute.

1613 update_type
1614 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
1615 that indicates how to handle the request if the attribute already exists in the
1616 attribute list.

1617 num_values
1618 The number of values that follow in the variable part of the argument list.

1619 ...
1620 The values to be added.

1621

1622 **Outputs.**

1623 attrs
1624 The attribute list is updated.

1625

1626 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
1627 failure value is returned.

1628 **Example.**

```
1629           #include "papi.h"  
1630  
1631           papi_attribute_t** attrs = NULL;  
1632           ...  
1633           papiAttributeAdd(&attrs,  
1634                            "job-name",  
1635                            PAPI_STRING,  
1636                            PAPI_EXCL,  
1637                            1,  
1638                            "My job" );  
1639           ...  
1640           papiAttributeListFree(attrs);  
1641
```

1642

1643 **See Also.** papiAttributeListFree, papiAttributeAddString, papiAttributeAddInteger,
1644 papiAttributeAddBoolean, papiAttributeAddRange, papiAttributeAddResolution,
1645 papiAttributeAddDatetime

1646 6.2. papiAttributeAddString

1647 **Description.** Add a string-valued attribute to an attribute list. Memory is allocated and
 1648 copies of the input arguments are created. It is the caller's responsibility to call
 1649 papiAttributeListFree when done with the attribute list.

1650 **Syntax.**

```
1651 papi_status_t papiAttributeAddString(  
1652     papi_attribute_t*** attrs,  
1653     const char* name,  
1654     const int update_type,  
1655     const int num_values,  
1656     ... );  
1657
```

1658

1659 **Inputs.**

1660 attrs

1661 Points to an attribute list. If a NULL value is passed, this function will allocate
 1662 the attribute list.

1663 name

1664 Points to the name of the attribute to add.

1665 update_type

1666 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 1667 that indicates how to handle the request if the attribute already exists in the
 1668 attribute list.

1669 num_values

1670 The number of values that follow in the variable part of the argument list.

1671 ...

1672 The values (char*) to be added.

1673

1674 **Outputs.**

1675 attrs

1676 The attribute list is updated.

1677

1678 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
1679 failure value is returned.

1680 **Example.**

```
1681 #include "papi.h"  
1682  
1683 papi_attribute_t** attrs = NULL;  
1684 ...  
1685 papiAttributeAddString(&attrs,  
1686                       "job-name",  
1687                       PAPI_EXCL,  
1688                       1,  
1689                       "My job" );  
1690 ...  
1691 papiAttributeListFree(attrs);  
1692
```

1693

1694 **See Also.** papiAttributeListFree, papiAttributeAdd

1695 6.3. papiAttributeAddInteger

1696 **Description.** Add an integer-valued attribute to an attribute list. Memory is allocated
1697 and copies of the input arguments are created. It is the caller's responsibility to call
1698 papiAttributeListFree when done with the attribute list.

1699 **Syntax.**

```
1700 papi_status_t papiAttributeAddInteger(  
1701     papi_attribute_t*** attrs,  
1702     const char* name,  
1703     const int update_type,  
1704     const int num_values,  
1705     ... );  
1706
```

1707

1708 **Inputs.**

1709 **attrs**
 1710 Points to an attribute list. If a NULL value is passed, this function will allocate
 1711 the attribute list.

1712 **name**
 1713 Points to the name of the attribute to add.

1714 **update_type**
 1715 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 1716 that indicates how to handle the request if the attribute already exists in the
 1717 attribute list.

1718 **num_values**
 1719 The number of values that follow in the variable part of the argument list.

1720 ...

1721 The values (int) to be added.

1722

1723 **Outputs.**

1724 **attrs**
 1725 The attribute list is updated.

1726

1727 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 1728 failure value is returned.

1729 **Example.**

```

1730            #include "papi.h"
1731
1732            papi_attribute_t** attrs = NULL;
1733            ...
1734            papiAttributeAddInteger (&attrs,
1735                                      "copies",
1736                                      PAPI_EXCL,
1737                                      1,
1738                                      3 );
1739            ...
1740            papiAttributeListFree (attrs);
1741
```

1742

1743 **See Also.** `papiAttributeListFree`, `papiAttributeAdd`

1744 **6.4. papiAttributeAddBoolean**

1745 **Description.** Add a boolean-valued attribute to an attribute list. Memory is allocated
1746 and copies of the input arguments are created. It is the caller's responsibility to call
1747 `papiAttributeListFree` when done with the attribute list.

1748 **Syntax.**

```
1749            papi_status_t papiAttributeAddBoolean(  
1750                        papi_attribute_t*** attrs,  
1751                        const char* name,  
1752                        const int update_type,  
1753                        const int num_values,  
1754                        ... );  
1755
```

1756

1757 **Inputs.**

1758 `attrs`

1759 Points to an attribute list. If a NULL value is passed, this function will allocate
1760 the attribute list.

1761 `name`

1762 Points to the name of the attribute to add.

1763 `update_type`

1764 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
1765 that indicates how to handle the request if the attribute already exists in the
1766 attribute list.

1767 `num_values`

1768 The number of values that follow in the variable part of the argument list.

1769 `...`

1770 The values (0 or 1) to be added.

1771

1772 **Outputs.**

1773 attrs

1774 The attribute list is updated.

1775

1776 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 1777 failure value is returned.

1778 **Example.**

```

1779       #include "papi.h"
1780
1781       papi_attribute_t** attrs = NULL;
1782       ...
1783       papiAttributeAddBoolean (&attrs,
1784                               "color-supported",
1785                               PAPI_EXCL,
1786                               1,
1787                               1 );
1788       ...
1789       papiAttributeListFree (attrs);
1790
```

1791

1792 **See Also.** papiAttributeListFree, papiAttributeAdd1793 **6.5. papiAttributeAddRange**

1794 **Description.** Add a range-valued attribute to an attribute list. Memory is allocated and
 1795 copies of the input arguments are created. It is the caller's responsibility to call
 1796 papiAttributeListFree when done with the attribute list.

1797 **Syntax.**

```

1798       papi_status_t papiAttributeAddRange (
1799                    papi_attribute_t*** attrs,
1800                    const char* name,
1801                    const int update_type,
1802                    const int lower,
1803                    const int upper );
1804
```

1805

1806 **Inputs.**

1807 attrs
1808 Points to an attribute list. If a NULL value is passed, this function will allocate
1809 the attribute list.

1810 name
1811 Points to the name of the attribute to add.

1812 update_type
1813 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
1814 that indicates how to handle the request if the attribute already exists in the
1815 attribute list.

1816 lower
1817 The lower range value.

1818 upper
1819 The upper range value.

1820

1821 **Outputs.**

1822 attrs
1823 The attribute list is updated.

1824

1825 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
1826 failure value is returned.

1827 **Example.**

```
1828            #include "papi.h"  
1829  
1830            papi_attribute_t** attrs = NULL;  
1831            ...  
1832            papiAttributeAddRange (&attrs,  
1833                                    "job-k-octets-supported",  
1834                                    PAPI_EXCL,  
1835                                    1,  
1836                                    100000 );  
1837            ...  
1838            papiAttributeListFree (attrs);  
1839
```

1840

1841 **See Also.** `papiAttributeListFree`

1842 **6.6. papiAttributeAddResolution**

1843 **Description.** Add a resolution-valued attribute to an attribute list. Memory is allocated
 1844 and copies of the input arguments are created. It is the caller's responsibility to call
 1845 `papiAttributeListFree` when done with the attribute list.

1846 **Syntax.**

```
1847 papi_status_t papiAttributeAddRange(
1848     papi_attribute_t*** attrs,
1849     const char* name,
1850     const int update_type,
1851     const int xres,
1852     const int yres );
1853
```

1854

1855 **Inputs.**

1856 `attrs`

1857 Points to an attribute list. If a NULL value is passed, this function will allocate
 1858 the attribute list.

1859 `name`

1860 Points to the name of the attribute to add.

1861 `update_type`

1862 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 1863 that indicates how to handle the request if the attribute already exists in the
 1864 attribute list.

1865 `xres`

1866 The X-axis resolution value.

1867 `yres`

1868 The Y-axis resolution value.

1869

1870 **Outputs.**

1871 attrs

1872 The attribute list is updated.

1873

1874 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
1875 failure value is returned.

1876 **Example.**

```
1877           #include "papi.h"  
1878  
1879           papi_attribute_t** attrs = NULL;  
1880           ...  
1881           papiAttributeAddResolution(&attrs,  
1882                                     "printer-resolution",  
1883                                     PAPI_EXCL,  
1884                                     300,  
1885                                     300 );  
1886           ...  
1887           papiAttributeListFree(attrs);  
1888
```

1889

1890 **See Also.** papiAttributeListFree

1891 **6.7. papiAttributeAddDatetime**

1892 **Description.** Add a date/time-valued attribute to an attribute list. Memory is allocated
1893 and copies of the input arguments are created. It is the caller's responsibility to call
1894 papiAttributeListFree when done with the attribute list.

1895 **Syntax.**

```
1896           papi_status_t papiAttributeAddDatetime(  
1897                 papi_attribute_t*** attrs,  
1898                 const char* name,  
1899                 const int update_type,  
1900                 const time_t* date_time );  
1901
```

1902

1903 **Inputs.**

1904 `attrs`
 1905 Points to an attribute list. If a NULL value is passed, this function will allocate
 1906 the attribute list.

1907 `name`
 1908 Points to the name of the attribute to add.

1909 `update_type`
 1910 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 1911 that indicates how to handle the request if the attribute already exists in the
 1912 attribute list.

1913 `date_time`
 1914 The date/time value.

1915

1916 **Outputs.**

1917 `attrs`
 1918 The attribute list is updated.

1919

1920 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 1921 failure value is returned.

1922 **Example.**

```

1923            #include "papi.h"
1924
1925            papi_attribute_t** attrs = NULL;
1926            time_t date_time
1927            ...
1928            time(&date_time);
1929            papiAttributeAddDatetime(&attrs,
1930                                      "date-time-at-creation",
1931                                      PAPI_EXCL,
1932                                      &date_time );
1933            ...
1934            papiAttributeListFree(attrs);
1935
```

1936

1937 **See Also.** `papiAttributeListFree`

1938 6.8. papiAttributeListFree

1939 **Description.** Frees an attribute list.

1940 **Syntax.**

```
1941 void papiAttributeListFree(  
1942     const papi_attribute_t** attrs );  
1943
```

1944

1945 **Inputs.**

1946 attrs

1947 Attribute list to be freed.

1948

1949 **Outputs.** none

1950 **Returns.** none

1951 **Example.**

```
1952 #include "papi.h"  
1953  
1954 papi_attribute_t** attrs = NULL;  
1955 ...  
1956 papiAttributeAddString(&attrs,  
1957     "job-name",  
1958     PAPI_EXCL,  
1959     1,  
1960     "My job" );  
1961 ...  
1962 papiAttributeListFree(attrs);  
1963
```

1964

1965 **See Also.** papiAttributeAddString, etc.

1966 6.9. papiAttributeListFind

1967 **Description.** Find an attribute in an attribute list.

1968 **Syntax.**

```
1969 papi_attribute_t* papiAttributeListFind(  
1970     const papi_attribute_t** attrs,
```

```

1971         const char*         name );
1972

```

1973

1974 **Inputs.**1975 `attrs`

1976 Attribute list to be searched.

1977 `name`

1978 Pointer to the name of the attribute to find.

1979

1980 **Outputs.** none

1981 **Returns.** Pointer to the found attribute. NULL indicates that the specified attribute was
 1982 not found

1983 **Example.**

```

1984 #include "papi.h"
1985
1986 papi_attribute_t** attrs = NULL;
1987 papi_attribute_t* attr = NULL;
1988 ...
1989 attr = papiAttributeListFind(&attrs,
1990                             "job-name" );
1991
1992 if (attr != NULL)
1993 {
1994     /* process the attribute */
1995     ...
1996 }
1997 ...
1998 papiAttributeListFree(attrs);

```

1999

2000 **See Also.** `papiAttributeListGetNext`

2001 6.10. `papiAttributeListGetNext`

2002 **Description.** Get the next attribute in an attribute list.

2003 **Syntax.**

```

2004 papi_attribute_t* papiAttributeListGetNext(
2005     const papi_attribute_t** attrs,

```

```

2006                 void**                 iterator );
2007

```

2008

2009 **Inputs.**

2010 attrs

2011 Attribute list to be used.

2012 iterator

2013 Pointer to an opaque (void*) iterator. This should be NULL to find the first
 2014 attribute and then passed in unchanged on subsequent calls to this function.

2015

2016 **Outputs.** none

2017 **Returns.** Pointer to the found attribute. NULL indicates that the end of the attribute list
 2018 was reached.

2019 **Example.**

```

2020 #include "papi.h"
2021
2022 papi_attribute_t** attrs = NULL;
2023 papi_attribute_t* attr = NULL;
2024 void* iterator = NULL;
2025 ...
2026 attr = papiAttributeListGetNext(&attrs,
2027                               &iterator );
2028 while (attr != NULL)
2029 {
2030     /* process this attribute */
2031     ...
2032     attr = papiAttributeListGetNext(&attrs,
2033                                   &iterator );
2034 }
2035 ...
2036 papiAttributeListFree(attrs);
2037

```

2038

2039 **See Also.** papiAttributeListFind

2040 Chapter 7. Job API

2041 7.1. papiJobSubmit

2042 **Description.** Submits a print job having the specified attributes to the specified printer.

2043 **Syntax.**

```
2044 papi_status_t papiJobSubmit(  
2045             papi_service_t      handle,  
2046             const char*         printer_name,  
2047             const papi_attribute_t** job_attributes,  
2048             const papi_job_ticket_t* job_ticket,  
2049             const char**        file_names,  
2050             papi_job_t**        job );  
2051
```

2052
2053 **Inputs.**

2054 handle
2055 Handle to the print service to use.

2056 printer_name
2057 Pointer to the name of the printer to which the job is to be submitted.

2058 job_attributes
2059 (optional) The list of attributes describing the job and how it is to be printed. If
2060 options are specified here and also in the job ticket data, the value specified here
2061 takes precedence. If this is NULL then only default attributes and (optionally) a
2062 job ticket is submitted with the job.

2063 job_ticket
2064 (optional) Pointer to structure specifying the job ticket. If this argument is NULL,
2065 then no job ticket is used with the job.

2066 file_names
2067 NULL terminated list of pointers to names of files to print.

2068

2069

Outputs.

2070

job

2071

The resulting job object representing the submitted job.

2072

2073

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

2074

2075

Example.

2076

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

2077

2078

```
papi_status_t status;
```

2079

```
papi_service_t handle = NULL;
```

2080

```
const char* printer = "my-printer";
```

2081

```
const papi_attribute_t** attrs = NULL;
```

2082

```
const papi_job_ticket_t* ticket = NULL;
```

2083

```
const char* files[] = { "/etc/motd", NULL };
```

2084

```
papi_job_t* job = NULL;
```

2085

2086

```
status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
```

2087

```
        PAPI_ENCRYPT_IF_REQUESTED, NULL);
```

2088

```
if (status != PAPI_OK)
```

2089

```
{
```

2090

```
    /* handle the error */
```

2091

```
    ...
```

2092

```
}
```

2093

2094

```
papiAttributeAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
```

2095

```
        PAPI_STRING, 1, "test job");
```

2096

```
papiAttributeAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
```

2097

```
        PAPI_INTEGER, 1, 4);
```

2098

2099

```
status = papiJobSubmit(handle,
```

2100

```
        printer,
```

2101

```
        attrs,
```

2102

```
        ticket,
```

2103

```
        files,
```

2104

```
        &job);
```

2105

```
if (status != PAPI_OK)
```

2106

```
{
```

2107

```
    fprintf(stderr, "papiJobSubmit failed: %s\n",
```

2108

```
            papiStatusString(status));
```

2109

```
    ...
```

2110

```
}
```

2111

2112

```
if (job != NULL)
```

2113

```
{
```

2114

```
    /* look at the job object (maybe get the id) */
```

2115

```
    papiJobFree(job);
```

2116

```
}
```

2117

```
papiServiceDestroy(handle);
```

2118

2119

2120

2121

2122 **See Also.** papiJobValidate, papiJobFree2123 **7.2. papiJobValidate**

2124 **Description.** Validates the specified job attributes against the specified printer. This
 2125 function can be used to validate the capability of a print object to accept a specific
 2126 combination of attributes.

2127 **Syntax.**

```

2128       papi_status_t papiJobValidate(
2129                    papi_service_t         handle,
2130                    const char*            printer_name,
2131                    const papi_attribute_t** job_attributes,
2132                    const papi_job_ticket_t* job_ticket,
2133                    const char**          file_names,
2134                    papi_job_t**          job );
2135

```

2136

2137 **Inputs.**

2138 handle

2139 Handle to the print service to use.

2140 printer_name

2141 Pointer to the name of the printer against which the job is to be validated.

2142 job_attributes

2143 (optional) The list of attributes describing the job and how it is to be printed. If
 2144 options are specified here and also in the job ticket data, the value specified here
 2145 takes precedence. If this is NULL then only default attributes and (optionally) a
 2146 job ticket is submitted with the job.

2147 job_ticket

2148 (optional) Pointer to structure specifying the JDF job ticket. If this argument is
 2149 NULL, then no job ticket is used with the job.

2150 file_names
 2151 NULL terminated list of pointers to names of files to validate.

2152

2153 **Outputs.**

2154 job

2155 The resulting job object representing what would be the submitted job.

2156

2157 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 2158 failure value is returned.

2159 **Example.**

```

2160 #include "papi.h"
2161
2162 papi_status_t status;
2163 papi_service_t handle = NULL;
2164 const char* printer = "my-printer";
2165 const papi_attribute_t** attrs = NULL;
2166 const papi_job_ticket_t* ticket = NULL;
2167 const char* files[] = { "/etc/motd", NULL };
2168 papi_job_t* job = NULL;
2169
2170 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
2171                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
2172 if (status != PAPI_OK)
2173 {
2174     /* handle the error */
2175     ...
2176 }
2177
2178 papiAttributeAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
2179                      PAPI_STRING, 1, "test job");
2180 papiAttributeAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
2181                       PAPI_INTEGER, 1, 4);
2182
2183 status = papiJobValidate(handle,
2184                        printer,
2185                        attrs,
2186                        ticket,
2187                        files,
2188                        &job);
2189 if (status != PAPI_OK)
2190 {
2191     fprintf(stderr, "papiJobValidate failed: %s\n",
2192           papiStatusString(status));
2193     ...
2194 }
2195
2196 if (job != NULL)
2197 {
2198     ...
2199     papiJobFree(job);
2200 }

```



```

2201
2202     papiServiceDestroy(handle);
2203

```

2204

2205 **See Also.** papiJobSubmit, papiJobFree

2206 7.3. papiJobQuery

2207 **Description.** Queries some or all the attributes of the specified job object.

2208 **Syntax.**

```

2209     papi_status_t papiJobQuery(
2210         papi_service_t     handle,
2211         const char*        printer_name,
2212         const int32_t      job_id,
2213         const char*        requested_attrs[],
2214         papi_job_t**       job );
2215

```

2216

2217 **Inputs.**

2218 handle

2219 Handle to the print service to use.

2220 printer_name

2221 Pointer to the name or URI of the printer to which the job was submitted.

2222 job_id

2223 The ID number of the job to be queried.

2224 requested_attrs

2225 NULL terminated array of attributes to be queried. If NULL is passed then all
 2226 available attributes are queried. (NOTE: The job may return more attributes than
 2227 you requested. This is merely an advisory request that may reduce the amount of
 2228 data returned if the printer/server supports it.)

2229

2230 **Outputs.**

2231 job

2232 The returned job object containing the requested attributes.

2233

2234 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
2235 failure value is returned.

2236 **Example.**

```

2237 #include "papi.h"
2238
2239 papi_status_t status;
2240 papi_service_t handle = NULL;
2241 const char* printer_name = "my-printer";
2242 papi_job_t* job = NULL;
2243 int32_t job_id = 12;
2244 const char* job_attrs[] =
2245 {
2246     "job-id",
2247     "job-name",
2248     "job-originating-user-name",
2249     "job-state",
2250     "job-state-reasons",
2251     NULL
2252 };
2253 ...
2254 status = papiServiceCreate(&handle,
2255                             NULL,
2256                             NULL,
2257                             NULL,
2258                             NULL,
2259                             PAPI_ENCRYPT_NEVER,
2260                             NULL);
2261 if (status != PAPI_OK)
2262 {
2263     /* handle the error */
2264     ...
2265 }
2266
2267 status = papiJobQuery(handle,
2268                       printer_name,
2269                       job_id,
2270                       job_attrs,
2271                       &job);
2272 if (status != PAPI_OK)
2273 {
2274     /* handle the error */
2275     fprintf(stderr, "papiJobQuery failed: %s\n",
2276            papiServiceGetStatusMessage(handle));
2277     ...
2278 }
2279
2280 if (job != NULL)
2281 {
2282     /* process the job */
2283     ...
2284     papiJobFree(job);
2285 }
2286

```

```
2287     papiServiceDestroy(handle);
2288
```

2289

2290 **See Also.** papiJobFree, papiPrinterListJobs

2291 7.4. papiJobCancel

2292 **Description.** Cancel the specified print job.

2293 **Syntax.**

```
2294     papi_status_t papiJobCancel(
2295         papi_service_t     handle,
2296         const char*         printer_name,
2297         const int32_t       job_id );
2298
```

2299

2300 **Inputs.**

2301 handle

2302 Handle to the print service to use.

2303 printer_name

2304 Pointer to the name or URI of the printer to which the job was submitted.

2305 job_id

2306 The ID number of the job to be cancelled.

2307

2308 **Outputs.** none

2309 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
2310 failure value is returned.

2311 **Example.**

```
2312     #include "papi.h"
2313
2314     papi_status_t status;
2315     papi_service_t handle = NULL;
2316     const char* printer_name = "my-printer";
2317     int32_t job_id = 12;
2318     ...
```

```

2319     status = papiServiceCreate(&handle,
2320                               NULL,
2321                               NULL,
2322                               NULL,
2323                               NULL,
2324                               PAPI_ENCRYPT_NEVER,
2325                               NULL);
2326
2327     if (status != PAPI_OK)
2328     {
2329         /* handle the error */
2330         ...
2331     }
2332
2333     status = papiJobCancel(handle,
2334                           printer_name,
2335                           job_id);
2336
2337     if (status != PAPI_OK)
2338     {
2339         /* handle the error */
2340         fprintf(stderr, "papiJobCancel failed: %s\n",
2341                papiServiceGetStatusMessage(handle));
2342         ...
2343     }
2344
2345     papiServiceDestroy(handle);

```

2345

2346 **See Also.** `papiPrinterListJobs`, `papiPrinterPurgeJobs`2347

7.5. papiJobHold

2348 **Description.** Holds the specified print job and prevents it from being scheduled for
 2349 printing. This operation is optional and may not be supported by all printers/servers.
 2350 Use `papiJobRelease` to undo the effects of this operation, or specify the `hold_until`
 2351 argument to automatically release the job at a specific time.

2352 **Syntax.**

```

2353 papi_status_t papiJobHold(
2354                 papi_service_t    handle,
2355                 const char*        printer_name,
2356                 const int32_t      job_id,
2357                 const char*        hold_until,
2358                 const time_t*      hold_until_time );
2359

```

2360

2361 **Inputs.**

2362 handle

2363 Handle to the print service to use.

2364 printer_name

2365 Pointer to the name or URI of the printer to which the job was submitted.

2366 job_id

2367 The ID number of the job to be held.

2368 hold_until

2369 (optional) Specifies the time when the job will be automatically released for

2370 printing. If NULL, the job is held until explicitly released by calling

2371 papiJobRelease. If specified, the value must be one of the strings "indefinite"

2372 (same effect as passing NULL), "day-time", "evening", "night", "weekend",

2373 "second-shift", "third-shift", or "timed". For values other than "indefinite" and

2374 "timed", the printer/server must define exact times associated with these values

2375 and it may make these associations configurable. If "timed" is specified, then the

2376 hold_until_time argument is used.

2377 hold_until_time

2378 (optional) Specifies the time when the job will be automatically released for

2379 printing. This argument is ignored unless "timed" is passed as the hold_until

2380 argument.

2381

2382 **Outputs.** none

2383 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate

2384 failure value is returned.

2385 **Example.**

2386 #include "papi.h"

2387

2388 papi_status_t status;

2389 papi_service_t handle = NULL;

2390 const char* printer_name = "my-printer";

2391 int32_t job_id = 12;

2392 ...

2393 status = papiServiceCreate(&handle,

2394 NULL,

2395 NULL,

2396 NULL,

2397 NULL,

2398 PAPI_ENCRYPT_NEVER,

```

2399                                     NULL);
2400     if (status != PAPI_OK)
2401     {
2402         /* handle the error */
2403         ...
2404     }
2405
2406     status = papiJobHold(handle,
2407                         printer_name,
2408                         job_id,
2409                         NULL,
2410                         NULL);
2411     if (status != PAPI_OK)
2412     {
2413         /* handle the error */
2414         fprintf(stderr, "papiJobHold failed: %s\n",
2415                papiServiceGetStatusMessage(handle));
2416         ...
2417     }
2418
2419     papiServiceDestroy(handle);
2420

```

2421

2422 **See Also.** `papiJobRelease`2423

7.6. `papiJobRelease`

2424 **Description.** Releases the specified print job, allowing it to be scheduled for printing.
 2425 This operation is optional and may not be supported by all printers/servers, but it must
 2426 be supported if `papiJobHold` is supported.

2427 **Syntax.**

```

2428 papi_status_t papiJobRelease(
2429             papi_service_t      handle,
2430             const char*         printer_name,
2431             const int32_t       job_id );
2432

```

2433

2434 **Inputs.**2435 `handle`

2436 Handle to the print service to use.

2437 `printer_name`

2438 Pointer to the name or URI of the printer to which the job was submitted.

2439 job_id
 2440 The ID number of the job to be released.

2441

2442 **Outputs.** none

2443 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 2444 failure value is returned.

2445 **Example.**

```

2446           #include "papi.h"
2447
2448           papi_status_t status;
2449           papi_service_t handle = NULL;
2450           const char* printer_name = "my-printer";
2451           int32_t job_id = 12;
2452           ...
2453           status = papiServiceCreate(&handle,
2454                                      NULL,
2455                                      NULL,
2456                                      NULL,
2457                                      NULL,
2458                                      PAPI_ENCRYPT_NEVER,
2459                                      NULL);
2460
2461           if (status != PAPI_OK)
2462           {
2463               /* handle the error */
2464               ...
2465           }
2466
2467           status = papiJobRelease(handle,
2468                                   printer_name,
2469                                   job_id);
2470
2471           if (status != PAPI_OK)
2472           {
2473               /* handle the error */
2474               fprintf(stderr, "papiJobRelease failed: %s\n",
2475                        papiServiceGetStatusMessage(handle));
2476               ...
2477           }
2478           papiServiceDestroy(handle);

```

2479

2480 **See Also.** papiJobHold

2481 7.7. papiJobRestart

2482 **Description.** Restarts a job that was retained after processing. If and how a job is
 2483 retained after processing is implementation-specific and is not covered by this API.
 2484 This operation is optional and may not be supported by all printers/servers.

2485 **Syntax.**

```

2486           papi_status_t papiJobRestart(
2487                           papi_service_t        handle,
2488                           const char*           printer_name,
2489                           const int32_t         job_id );
2490

```

2491

2492 **Inputs.**

2493 handle

2494 Handle to the print service to use.

2495 printer_name

2496 Pointer to the name or URI of the printer to which the job was submitted.

2497 job_id

2498 The ID number of the job to be restarted.

2499

2500 **Outputs.** none

2501 **Returns.** If successful, a value of PAPI_OK is returned. Otherwise an appropriate
 2502 failure value is returned.

2503 **Example.**

```

2504           #include "papi.h"
2505
2506           papi_status_t status;
2507           papi_service_t handle = NULL;
2508           const char* printer_name = "my-printer";
2509           int32_t job_id = 12;
2510           ...
2511           status = papiServiceCreate(&handle,
2512                                    NULL,
2513                                    NULL,
2514                                    NULL,
2515                                    NULL,
2516                                    PAPI_ENCRYPT_NEVER,
2517                                    NULL);
2518
2519           if (status != PAPI_OK)
2520           {
2521               /* handle the error */
2522               ...
2523           }
2524
2525           status = papiJobRestart(handle,
2526                                   printer_name,

```



```

2526             job_id);
2527 if (status != PAPI_OK)
2528 {
2529     /* handle the error */
2530     fprintf(stderr, "papiJobRestart failed: %s\n",
2531            papiServiceGetStatusMessage(handle));
2532     ...
2533 }
2534
2535 papiServiceDestroy(handle);
2536

```

2537

2538 **See Also.** `papiPrinterListJobs`

2539 7.8. `papiJobFree`

2540 **Description.** Free a job object.2541 **Syntax.**

```

2542 void papiJobFree (
2543             papi_job_t*      job );
2544

```

2545

2546 **Inputs.**2547 `job`

2548 Pointer to the printer object to free.

2549

2550 **Outputs.** none2551 **Returns.** none2552 **Example.**

```

2553 #include "papi.h"
2554
2555 papi_status_t status;
2556 papi_service_t handle = NULL;
2557 const char* printer_name = "my-printer";
2558 papi_job_t* job = NULL;
2559 ...
2560 status = papiServiceCreate(&handle,
2561                          NULL,
2562                          NULL,
2563                          NULL,
2564                          NULL,

```

```

2565                                     PAPI_ENCRYPT_NEVER,
2566                                     NULL);
2567     if (status != PAPI_OK)
2568     {
2569         /* handle the error */
2570         ...
2571     }
2572
2573     status = papiJobQuery(handle,
2574                         printer_name,
2575                         12,
2576                         &job);
2577     if (status != PAPI_OK)
2578     {
2579         /* handle the error */
2580         fprintf(stderr, "papiJobQuery failed: %s\n",
2581                papiServiceGetStatusMessage(handle));
2582         ...
2583     }
2584
2585     if (job != NULL)
2586     {
2587         /* process the job object */
2588         ...
2589         papiJobFree(job);
2590     }
2591
2592     papiServiceDestroy(handle);
2593

```

2594

2595 **See Also.** papiJobQuery

2596 7.9. papiJobListFree

2597 **Description.** Free a list of job objects.2598 **Syntax.**

```

2599 void papiJobListFree(
2600                 papi_job_t** jobs );
2601

```

2602

2603 **Inputs.**

2604 jobs

2605 Pointer to the printer object list to free.

2606

2607 **Outputs.** none

2608 **Returns.** none

2609 **Example.**

```

2610 #include "papi.h"
2611
2612 papi_status_t status;
2613 papi_service_t handle = NULL;
2614 const char* printer_name = "my-printer";
2615 papi_printer_t** printers = NULL;
2616 ...
2617 status = papiServiceCreate(&handle,
2618                             NULL,
2619                             NULL,
2620                             NULL,
2621                             NULL,
2622                             PAPI_ENCRYPT_NEVER,
2623                             NULL);
2624
2625 if (status != PAPI_OK)
2626 {
2627     /* handle the error */
2628     ...
2629 }
2630
2631 status = papiPrinterListJobs(handle,
2632                             printer_name,
2633                             NULL,
2634                             0, 0, 0,
2635                             &jobs);
2636
2637 if (status != PAPI_OK)
2638 {
2639     /* handle the error */
2640     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
2641             papiServiceGetStatusMessage(handle));
2642     ...
2643 }
2644
2645 if (jobs != NULL)
2646 {
2647     /* process the job objects */
2648     ...
2649     papiJobListFree(jobs);
2650 }
2651
2652 papiServiceDestroy(handle);

```

2652

2653 **See Also.** `papiPrinterListJobs`

2654 Chapter 8. Miscellaneous API

2655 8.1. papiStatusString

2656 **Description.** Get a status string for the specified `papi_status_t`. The status message
2657 returned from this function may be less detailed than the status message returned from
2658 `papiServiceGetStatusMessage` (if the print service supports returning more detailed
2659 error messages).

2660 The returned message will be localized in the language of the submitter of the
2661 requestor.

2662 **Syntax.**

```
2663 char* papiStatusString(  
2664     const papi_status_t status );  
2665
```

2666

2667 **Inputs.**

2668 status

2669 The status value to convert to a status string.

2670

2671 **Outputs.** none

2672 **Returns.** If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate
2673 failure value is returned.

2674 **Example.**

```
2675 #include "papi.h"  
2676  
2677 papi_status_t status;  
2678 ...  
2679 fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));  
2680
```

2681

2682 **See Also.** `papiServiceGetStatusMessage`

2683 **Chapter 9. Attributes**

2684 * *ISSUE: Waiting for reference to single document from Tom H.*
2685

2686 **9.1. Extension Attributes**

2687 The following attributes are not currently defined by IPP, but may be used with this
2688 API.

2689 **9.1.1. job-ticket-formats-supported**

2690 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats that
2691 are supported by the printer. If this attribute is not present, it is assumed that the printer
2692 does not support any job ticket formats.

2693 * *ISSUE: I took the following required attr lists directly from IPP RFC 2911 to use as a starting point. We*
2694 *probably want to add/delete attrs from the lists.*
2695

2696 **9.2. Required Job Attributes**

2697 The following job attributes *must* be supported to comply with this API standard.
2698 These attributes may be supported by the underlying print server directly, or they may
2699 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

2700 time-at-completed

2701 **9.3. Required Printer Attributes**

2702 The following printer attributes *must* be supported to comply with this API standard.
2703 These attributes may be supported by the underlying print server directly, or they may
2704 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

2705

2706 **Appendix A. Change History**

2707 **Version 0.3 (June 24, 2002).**

- 2708 • Converted to DocBook format from Microsoft Word
- 2709 • Major rewrite, including:
 - 2710 • Changed how printer names are described in "Model/Printer"
 - 2711 • Changed fixed length strings to pointers in numerous structures/sections
 - 2712 • Redefined attribute/value structures and associated API descriptions
 - 2713 • Changed list/query functions to return "objects"
 - 2714 • Rewrote "Attributes API" chapter
 - 2715 • Changed many function definitions to pass NULL-terminated arrays of pointers
 - 2716 instead of a separate count argument
 - 2717 • Changed papiJobSubmit to take an attribute list structure as input instead of a
 - 2718 formatted string

2719

2720

2721 **Version 0.2 (April 17, 2002).**

- 2722 • Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)
- 2723 • Filled in "Encryption" section and added information about encryption in "Object
- 2724 Identification" section
- 2725 • Added "short_name" field in "Object Identification" section
- 2726 • Added "Job Ticket (papi_job_ticket_t)" section
- 2727 • Added papiPrinterPause
- 2728 • Added papiPrinterResume
- 2729 • Added papiPurgeJobs
- 2730 • Added optional job_ticket argument to papiJobSubmit
- 2731 • Added optional passing of filenames by URI to papiJobSubmit
- 2732 • Added papiHoldJob
- 2733 • Added papiReleaseJob

Appendix A. Change History

2734 • Added papiRestartJob

2735

2736 **Version 0.1 (April 3, 2002).**

2737 • Original draft version

2738

2739

2740

2741

2742

2743

<i>End of Document</i>
