

1

2

Open Standard Print API (PAPI)

3

Version 0.6 (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

11

12 **Open Standard Print API (PAPI): Version 0.6 (DRAFT)**

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

14 Version 0.6 (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.

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

Table of Contents

1. Introduction.....	1
2. Print System Model	2
2.1. Introduction.....	2
2.2. Model.....	2
2.2.1. Print Service	2
2.2.2. Printer	2
2.2.3. Job.....	3
2.2.4. Document	3
2.3. Security.....	3
2.3.1. Authentication	3
2.3.2. Authorization.....	3
2.3.3. Encryption.....	3
3. Common Structures	4
3.1. Conventions.....	4
3.2. Service Object (papi_service_t)	4
3.3. Attributes and Values	4
3.4. Job Object (papi_job_t).....	5
3.5. Stream Object (papi_stream_t).....	5
3.6. Printer Object (papi_printer_t).....	5
3.7. Job Ticket (papi_job_ticket_t).....	5
3.8. Status (papi_status_t).....	6
3.9. List Filter (papi_filter_t).....	7
4. Service API	8
4.1. papiServiceCreate	8
4.2. papiServiceDestroy.....	9
4.3. papiServiceSetUsername	10
4.4. papiServiceSetPassword	12
4.5. papiServiceSetEncryption.....	13
4.6. papiServiceSetAuthCB.....	14
4.7. papiServiceSetAppData	15
4.8. papiServiceGetServicename.....	16
4.9. papiServiceGetUsername	17
4.10. papiServiceGetPassword	18
4.11. papiServiceGetEncryption.....	19
4.12. papiServiceGetAppData	19
4.13. papiServiceGetStatusMessage	20
5. Printer API.....	22
5.1. Usage	22
5.2. papiPrintersList.....	22
5.3. papiPrinterQuery.....	24
5.4. papiPrinterModify	26
5.5. papiPrinterPause.....	27
5.6. papiPrinterResume	29
5.7. papiPrinterPurgeJobs	30
5.8. papiPrinterListJobs	31
5.9. papiPrinterGetAttributeList.....	33
5.10. papiPrinterFree	34
5.11. papiPrinterListFree.....	35

67	6. Attributes API	37
68	6.1. papiAttributeListAdd	37
69	6.2. papiAttributeListAddString.....	38
70	6.3. papiAttributeListAddInteger.....	39
71	6.4. papiAttributeListAddBoolean	40
72	6.5. papiAttributeListAddRange	42
73	6.6. papiAttributeListAddResolution.....	43
74	6.7. papiAttributeListAddDatetime	44
75	6.8. papiAttributeListAddCollection.....	46
76	6.9. papiAttributeDelete.....	47
77	6.10. papiAttributeListGetValue.....	48
78	6.11. papiAttributeListGetString.....	49
79	6.12. papiAttributeListGetInteger.....	51
80	6.13. papiAttributeListGetBoolean	52
81	6.14. papiAttributeListGetRange	53
82	6.15. papiAttributeListGetResolution	54
83	6.16. papiAttributeListGetDatetime	55
84	6.17. papiAttributeListGetCollection	57
85	6.18. papiAttributeListFree.....	58
86	6.19. papiAttributeListFind	59
87	6.20. papiAttributeListGetNext.....	60
88	6.21. papiAttributeListFromString	61
89	6.22. papiAttributeListToString	62
90	7. Job API	64
91	7.1. papiJobSubmit.....	64
92	7.2. papiJobSubmitByReference	66
93	7.3. papiJobValidate.....	68
94	7.4. papiJobStreamOpen	69
95	7.5. papiJobStreamWrite	71
96	7.6. papiJobStreamClose	72
97	7.7. papiJobQuery	73
98	7.8. papiJobModify	75
99	7.9. papiJobCancel	76
100	7.10. papiJobHold	78
101	7.11. papiJobRelease	79
102	7.12. papiJobRestart	80
103	7.13. papiJobGetAttributeList	82
104	7.14. papiJobGetPrinterName	83
105	7.15. papiJobGetId	84
106	7.16. papiJobGetJobTicket.....	84
107	7.17. papiJobFree.....	85
108	7.18. papiJobListFree	86
109	8. Miscellaneous API	88
110	8.1. papiStatusString.....	88
111	9. Attributes	89
112	9.1. Extension Attributes.....	89
113	9.1.1. job-ticket-formats-supported	89
114	9.2. Required Job Attributes	89
115	9.3. Required Printer Attributes.....	89
116	9.4. IPP Attribute Type Mapping.....	90

117	A. Attribute List Text Representation	91
118	A.1. ABNF Definition	91
119	A.2. Examples	91
120	B. References	93
121	B.1. Internet Printing Protocol (IPP)	93
122	93
123	B.2. Job Ticket	93
124	93
125	B.3. Printer Working Group (PWG)	93
126	93
127	B.4. Other	93
128	93
129	C. Change History	94

130 Chapter 1. Introduction

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

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

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

140

141

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

144 Chapter 2. Print System Model

145 2.1. Introduction

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

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

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

156

157 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	

158

159

160 2.2. Model

161 The model on which the Open Standard Print API is derived from are the
162 semantics defined by the Internet Printing Protocol (IPP) standard. This is a fairly
163 simple model in terms of the number of object types. It is defined very clearly and
164 in detail in the IPP [RFC2911], Chapter 2
165 (<http://ietf.org/rfc/rfc2911.txt?number=2911>). See also other IPP-related
166 documents in Appendix B.

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

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

172 2.2.1. Print Service

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

177 2.2.2. Printer

178 Printer objects are the target of print job requests. A printer object may represent an
179 actual printer (if the printer itself supports PAPI), an object in a server representing
180 an actual printer, or an abstract object in a server (perhaps representing a pool or
181 class of printers). Printer objects are identified via one or more names which may be
182 short, local names (such as "prtr1") or longer global names (such as a URI like

183 "http://printserv.mycompany.com:631/printers/prtr1"). The PAPI implementation
 184 may detect and map short names to long global names in an implementation-
 185 specific way.

186 **2.2.3. Job**

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

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

193 **2.2.4. Document**

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

197 This version of PAPI does *NOT* support separate document objects, but they will
 198 probably be added in a future version. This might be done by adding new "Open
 199 job", "Add document", and "Close job" functions that will allow submitting a
 200 multiple document job and specifying separate attributes for each document.

201 **2.3. Security**

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

204 **2.3.1. Authentication**

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

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

211 **2.3.2. Authorization**

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

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

220 **2.3.3. Encryption**

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

226 Chapter 3. Common Structures

227 3.1. Conventions

228

229 • All "char*" variables and fields are pointers to standard C/C++ NULL-terminated
230 strings. It is assumed that these strings are all UTF-8 encoded characters strings.

231 • All pointer arrays (e.g. "char**") are assumed to be terminated by NULL pointers.
232 That is, the valid elements of the array are followed by an element containing a
233 NULL pointer that marks the end of the list.

234

235 3.2. Service Object (papi_service_t)

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

```
239 typedef void* papi_service_t;
```

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

```
243 typedef enum  
244 {  
245     PAPI_ENCRYPT_IF_REQUESTED, /* Encrypt if requested (TLS upgrade) */  
246     PAPI_ENCRYPT_NEVER,      /* Never encrypt */  
247     PAPI_ENCRYPT_REQUIRED,  /* Encryption is required (TLS upgrade) */  
248     PAPI_ENCRYPT_ALWAYS,    /* Always encrypt (SSL) */  
249 } papi_encryption_t;
```

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

253 3.3. Attributes and Values

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

```
256 /* Attribute Type */  
257 typedef enum  
258 {  
259     PAPI_STRING,  
260     PAPI_INTEGER,  
261     PAPI_BOOLEAN,  
262     PAPI_RANGE,  
263     PAPI_RESOLUTION,  
264     PAPI_DATETIME,  
265     PAPI_COLLECTION  
266 } papi_attribute_value_type_t;
```

```
268 /* Resolution units */  
269 typedef enum  
270 {  
271     PAPI_RES_PER_INCH = 3,  
272     PAPI_RES_PER_CM  
273 } papi_res_t;
```

```
275 /* Boolean values */  
276 enum  
277 {  
278     PAPI_FALSE = 0,  
279     PAPI_TRUE = 1  
280 };
```

281

```

282 struct papi_attribute_str;
283
284 /* Attribute Value */
285 typedef union
286 {
287     char* string; /* PAPI_STRING value */
288
289     int integer; /* PAPI_INTEGER value */
290
291     char boolean; /* PAPI_BOOLEAN value */
292
293     struct /* PAPI_RANGE value */
294     {
295         int lower;
296         int upper;
297     } range;
298
299     struct /* PAPI_RESOLUTION value */
300     {
301         int xres;
302         int yres;
303         papi_res_t units;
304     } resolution;
305
306     time_t datetime; /* PAPI_DATETIME value */
307
308     struct papi_attribute_str**
309     collection; /* PAPI_COLLECTION value */
310 } papi_attribute_value_t;
311
312
313 /* Attribute and Values */
314 typedef struct papi_attribute_str
315 {
316     char* name; /* attribute name */
317     papi_attribute_value_type_t type; /* type of values */
318     papi_attribute_value_t** values; /* list of values */
319 } papi_attribute_t;
320
321 /* Attribute add flags */
322 #define PAPI_ATTR_APPEND 0x0001 /* Add values to attr */
323 #define PAPI_ATTR_REPLACE 0x0002 /* Delete existing
324 values then add new ones */
325 #define PAPI_ATTR_EXCL 0x0004 /* Fail if attr exists */

```

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

327 3.4. Job Object (papi_job_t)

328 This opaque structure is used as a "handle" to information associated with a job
329 object. This handle is returned in response to successful job query/list operations.
330 See the "papiJobGet*" functions to see what information can be retrieved from the
331 job object using the handle.

332 3.5. Stream Object (papi_stream_t)

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

335 3.6. Printer Object (papi_printer_t)

336 This opaque structure is used as a "handle" to information associated with a printer
337 object. This handle is returned in response to successful job query/list operations.
338 See the "papiPrinterGet*" functions to see what information can be retrieved from
339 the printer object using the handle.

340 3.7. Job Ticket (papi_job_ticket_t)

341 This is the structure used to pass a job ticket when submitting a print job.
342 Currently, Job Definition Format (JDF) is the only supported job ticket format. JDF

343 is an XML- based job ticket syntax. The JDF specification can be found at
 344 www.cip4.org.

```

345 /* Job Ticket Format */
346 typedef enum
347 {
348     PAPI_JT_FORMAT_JDF = 0,      /* Job Definition Format */
349     PAPI_JT_FORMAT_PWG = 1,     /* PWG Job Ticket Format */
350 } papi_jt_format_t;
351
352
353 /* Job Ticket */
354 typedef struct papi_job_ticket_s
355 {
356     papi_jt_format_t format;     /* Format of job ticket */
357     char* ticket_data;         /* Buffer containing the job
358                                ticket data. If NULL,
359                                uri must be specified */
360     char* file_name;           /* Name of the file containing
361                                the job ticket data. If
362                                ticket_data is specified, then
363                                uri is ignored. */
364 } papi_job_ticket_t;
  
```

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

372 3.8. Status (papi_status_t)

```

373 typedef enum
374 {
375     PAPI_OK = 0x0000,
376     PAPI_OK_SUBST,
377     PAPI_OK_CONFLICT,
378     PAPI_OK_IGNORED_SUBSCRIPTIONS,
379     PAPI_OK_IGNORED_NOTIFICATIONS,
380     PAPI_OK_TOO_MANY_EVENTS,
381     PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
382     PAPI_REDIRECTION_OTHER_SITE = 0x300,
383     PAPI_BAD_REQUEST = 0x0400,
384     PAPI_FORBIDDEN,
385     PAPI_NOT_AUTHENTICATED,
386     PAPI_NOT_AUTHORIZED,
387     PAPI_NOT_POSSIBLE,
388     PAPI_TIMEOUT,
389     PAPI_NOT_FOUND,
390     PAPI_GONE,
391     PAPI_REQUEST_ENTITY,
392     PAPI_REQUEST_VALUE,
393     PAPI_DOCUMENT_FORMAT,
394     PAPI_ATTRIBUTES,
395     PAPI_URI_SCHEME,
396     PAPI_CHARSET,
397     PAPI_CONFLICT,
398     PAPI_COMPRESSION_NOT_SUPPORTED,
399     PAPI_COMPRESSION_ERROR,
400     PAPI_DOCUMENT_FORMAT_ERROR,
401     PAPI_DOCUMENT_ACCESS_ERROR,
402     PAPI_ATTRIBUTES_NOT_SETTABLE,
403     PAPI_IGNORED_ALL_SUBSCRIPTIONS,
404     PAPI_TOO_MANY_SUBSCRIPTIONS,
405     PAPI_IGNORED_ALL_NOTIFICATIONS,
406     PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
407     PAPI_INTERNAL_ERROR = 0x0500,
408     PAPI_OPERATION_NOT_SUPPORTED,
409     PAPI_SERVICE_UNAVAILABLE,
410     PAPI_VERSION_NOT_SUPPORTED,
411     PAPI_DEVICE_ERROR,
412     PAPI_TEMPORARY_ERROR,
413     PAPI_NOT_ACCEPTING,
414     PAPI_PRINTER_BUSY,
415     PAPI_ERROR_JOB_CANCELLED,
416     PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
417     PAPI_PRINTER_IS_DEACTIVATED,
  
```

```

418     PAPI_BAD_ARGUMENT
419 } papi_status_t;
420

```

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

424 3.9. List Filter (papi_filter_t)

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

```

428 typedef enum
429 {
430     PAPI_FILTER_BITMASK = 0
431     /* future filter types may be added here */
432 } papi_filter_type_t;
433
434 typedef struct
435 {
436     papi_filter_type_t type; /* Type of filter specified */
437
438     union
439     {
440         unsigned int mask; /* PAPI_FILTER_BITMASK */
441
442         /* future filter types may be added here */
443     } u;
444 } papi_filter_t;
445

```

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

```

448 enum
449 {
450     PAPI_PRINTER_LOCAL = 0x0000, /* Local printer or class */
451     PAPI_PRINTER_CLASS = 0x0001, /* Printer class */
452     PAPI_PRINTER_REMOTE = 0x0002, /* Remote printer or class */
453     PAPI_PRINTER_BW = 0x0004, /* Can do B&W printing */
454     PAPI_PRINTER_COLOR = 0x0008, /* Can do color printing */
455     PAPI_PRINTER_DUPLEX = 0x0010, /* Can do duplexing */
456     PAPI_PRINTER_STAPLE = 0x0020, /* Can staple output */
457     PAPI_PRINTER_COPIES = 0x0040, /* Can do copies */
458     PAPI_PRINTER_COLLATE = 0x0080, /* Can collage copies */
459     PAPI_PRINTER_PUNCH = 0x0100, /* Can punch output */
460     PAPI_PRINTER_COVER = 0x0200, /* Can cover output */
461     PAPI_PRINTER_BIND = 0x0400, /* Can bind output */
462     PAPI_PRINTER_SORT = 0x0800, /* Can sort output */
463     PAPI_PRINTER_SMALL = 0x1000, /* Can do Letter/Legal/A4 */
464     PAPI_PRINTER_MEDIUM = 0x2000, /* Can do Tabloid/B/C/A3/A2 */
465     PAPI_PRINTER_LARGE = 0x4000, /* Can do D/E/A1/A0 */
466     PAPI_PRINTER_VARIABLE = 0x8000, /* Can do variable sizes */
467     PAPI_PRINTER_IMPLICIT = 0x10000, /* Implicit class */
468     PAPI_PRINTER_DEFAULT = 0x20000, /* Default printer on network */
469     PAPI_PRINTER_OPTIONS = 0xffffc /* ~(CLASS | REMOTE | IMPLICIT) */
470 };
471

```

472 Chapter 4. Service API

473 4.1. papiServiceCreate

474 Description

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

479 Syntax

480

```
481 papi_status_t papiServiceCreate(  
482     papi_service_t*      handle,  
483     const char*          service_name,  
484     const char*          user_name,  
485     const char*          password,  
486     const int (*authCB) (papi_service_t svc),  
487     const papi_encryption_t encryption,  
488     void*                app_data );  
489
```

490

491 Inputs

492

493 service_name

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

498 user_name

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

502 password

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

505 authCB

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

511 encryption

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

513 app_data

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

516

517 **Outputs**

518

519 handle

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

523

524 **Returns**

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

527 **Example**

528

```

529 #include "papi.h"
530
531 papi_status_t status;
532 papi_service_t handle = NULL;
533 const char* service_name = "ipp:/printserv:631";
534 const char* user_name = "pappy";
535 const char* password = "goober";
536 ...
537 status = papiServiceCreate(&handle,
538                          service_name,
539                          user_name,
540                          password,
541                          NULL,
542                          PAPI_ENCRYPT_IF_REQUESTED,
543                          NULL);
544
545 if (status != PAPI_OK)
546 {
547     /* handle the error */
548     fprintf(stderr, "papiServiceCreate failed: %s\n",
549           papiStatusString(status));
550     if (handle != NULL)
551     {
552         fprintf(stderr, "    details: %s\n",
553               papiServiceGetStatusMessage(handle));
554     }
555     ...
556 }
557 papiServiceDestroy(handle);
558

```

559

560 **See Also**

561 papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUsername,
562 papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB

563 4.2. papiServiceDestroy

564 **Description**

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

568 **Syntax**

569

```
570           void papiServiceDestroy(
571                 papi_service_t handle );
572
```

573

574 **Inputs**

575

576 handle

The print service handle to be destroyed.

578

579 **Outputs**

580 none

581 **Returns**

582 none

583 **Example**

584

```
585           #include "papi.h"
586
587           papi_status_t status;
588           papi_service_t handle = NULL;
589           const char* service_name = "ipp://printserv:631";
590           const char* user_name = "pappy";
591           const char* password = "goober";
592           ...
593           status = papiServiceCreate(&handle,
594                                     service_name,
595                                     user_name,
596                                     password,
597                                     NULL,
598                                     PAPI_ENCRYPT_IF_REQUESTED,
599                                     NULL);
600
601           if (status != PAPI_OK)
602           {
603                 /* handle the error */
604                 ...
605           }
606           ...
607           papiServiceDestroy(handle);
```

608

609 **See Also**

610 papiServiceCreate

611 **4.3. papiServiceSetUsername**

612 **Description**

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

616 **Syntax**

617


```

618     papi_status_t papiServiceSetUsername (
619         papi_service_t handle,
620         const char* user_name );
621

```

622

623 **Inputs**

624

625 **handle**

626 Handle to the print service to update.

627 **user_name**

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

631

632 **Outputs**

633 handle is updated.

634 **Returns**

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

637 **Example**

638

```

639 #include "papi.h"
640
641 papi_status_t status;
642 papi_service_t handle = NULL;
643 const char* user_name = "pappy";
644 ...
645 status = papiServiceCreate(&handle,
646     NULL,
647     NULL,
648     NULL,
649     NULL,
650     PAPI_ENCRYPT_IF_REQUESTED,
651     NULL);
652
653 if (status != PAPI_OK)
654 {
655     /* handle the error */
656     ...
657 }
658
659 status = papiServiceSetUsername(handle, user_name);
660 if (status != PAPI_OK)
661 {
662     /* handle the error */
663     fprintf(stderr, "papiServiceSetUsername failed: %s\n",
664         papiServiceGetStatusMessage(handle));
665     ...
666 }
667 ...
668 papiServiceDestroy(handle);

```

669

670 **See Also**

671 papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

672 **4.4. papiServiceSetPassword**673 **Description**

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

677 **Syntax**

678

```
679 papi_status_t papiServiceSetPassword(
680     papi_service_t handle,
681     const char* password );
682
```

683

684 **Inputs**

685

686 handle

687 Handle to the print service to update.

688 password

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

690

691 **Outputs**

692 handle is updated.

693 **Returns**

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

696 **Example**

697

```
698 #include "papi.h"
699
700 papi_status_t status;
701 papi_service_t handle = NULL;
702 const char* password = "goober";
703 ...
704 status = papiServiceCreate(&handle,
705     NULL,
706     NULL,
707     NULL,
708     NULL,
709     PAPI_ENCRYPT_IF_REQUESTED,
710     NULL);
711
712 if (status != PAPI_OK)
713 {
714     /* handle the error */
715     ...
716 }
717
718 status = papiServiceSetPassword(handle, password);
719 if (status != PAPI_OK)
720 {
721     /* handle the error */
722     fprintf(stderr, "papiServiceSetPassword failed: %s\n",
723         papiServiceGetStatusMessage(handle));
724     ...
725 }
```

```
726     papiServiceDestroy(handle);
727
```

728

729 **See Also**

730 papiServiceCreate, papiServiceSetUsername, papiServiceGetStatusMessage

731 **4.5. papiServiceSetEncryption**732 **Description**

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

734 **Syntax**

735

```
736     papi_status_t papiServiceSetEncryption(
737         papi_service_t handle,
738         const papi_encryption_t encryption );
739
```

740

741 **Inputs**

742

743 handle

744 Handle to the print service to update.

745 encryption

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

747

748 **Outputs**

749 handle is updated.

750 **Returns**751 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
752 value is returned.753 **Example**

754

```
755     #include "papi.h"
756
757     papi_status_t status;
758     papi_service_t handle = NULL;
759     ...
760     status = papiServiceCreate(&handle,
761                               NULL,
762                               NULL,
763                               NULL,
764                               NULL,
765                               PAPI_ENCRYPT_IF_REQUESTED,
766                               NULL);
767
768     if (status != PAPI_OK)
769     {
770         /* handle the error */
771         ...
772     }
773
774     status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
775     if (status != PAPI_OK)
```

```

775     {
776         /* handle the error */
777         fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
778                papiServiceGetStatusMessage(handle));
779         ...
780     }
781     ...
782     papiServiceDestroy(handle);
783

```

784

785 **See Also**

786 papiServiceCreate, papiServiceGetStatusMessage

787 **4.6. papiServiceSetAuthCB**788 **Description**

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

791 **Syntax**

792

```

793 papi_status_t papiServiceSetAuthCB(
794     papi_service_t handle,
795     const int (*authCB)(papi_service_t svc) );
796

```

797

798 **Inputs**

799

800 handle

801 Handle to the print service to update.

802 authCB

803 Points to a callback function to be used in authenticating the user to the print
804 service if no password was supplied (or user input is required). A NULL value
805 indicates that no callback should be made. The callback function should return
806 0 if the request is to be cancelled and non-zero if new authentication
807 information has been set.

808

809 **Outputs**

810 handle is updated.

811 **Returns**

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

814 **Example**

815

```

816 #include "papi.h"
817
818 extern int get_password(papi_service_t handle);
819 papi_status_t status;
820 papi_service_t handle = NULL;

```

```

821     ...
822     status = papiServiceCreate(&handle,
823                               NULL,
824                               NULL,
825                               NULL,
826                               NULL,
827                               PAPI_ENCRYPT_IF_REQUESTED,
828                               NULL);
829
830     if (status != PAPI_OK)
831     {
832         /* handle the error */
833         ...
834     }
835
836     status = papiServiceSetAuthCB(handle, get_password);
837     if (status != PAPI_OK)
838     {
839         /* handle the error */
840         fprintf(stderr, "papiServiceSetAuthCB failed: %s\n",
841                papiServiceGetStatusMessage(handle));
842         ...
843     }
844     ...
845     papiServiceDestroy(handle);

```

846

847

See Also

848

papiServiceCreate, papiServiceGetStatusMessage

849

4.7. papiServiceSetAppData

850

Description

851

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

852

853

854

Syntax

855

```

856     papi_status_t papiServiceSetAppData (
857         papi_service_t handle,
858         const void*   app_data );
859

```

860

861

Inputs

862

863 handle

Handle to the print service to update.

864

865 app_data

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

866

867

868

869

Outputs

870

handle is updated.

871

Returns

872

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

873

874

Example

875

876

```
#include "papi.h"

extern int get_password(papi_service_t handle);
papi_status_t status;
papi_service_t handle = NULL;
char* app_data = "some data";
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);

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

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

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

See Also

898

papiServiceCreate, papiServiceGetStatusMessage

4.8. papiServiceGetServicename

907

Description

908

Get the service name associated with the print service handle.

909

Syntax

910

911

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

912

913

914

915

Inputs

916

917

handle

918

Handle to the print service.

919

920

924

Outputs

925

none

926

Returns

927

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

928

Example

929

```

930      #include "papi.h"
931
932      papi_status_t status;
933      papi_service_t handle = NULL;
934      char* service_name = NULL;
935      ...
936      service_name = papiServiceGetServicename(handle);
937      if (service_name != NULL)
938      {
939          /* use the returned name */
940          ...
941      }
942      ...
943      papiServiceDestroy(handle);
944

```

945

946

See Also

947

papiServiceCreate

948 **4.9. papiServiceGetUsername**

949

Description

950

Get the user name associated with the print service handle.

951

Syntax

952

```

953      char* papiServiceGetUsername(
954          papi_service_t handle );
955

```

956

957

Inputs

958

959 handle

960

Handle to the print service.

961

962

Outputs

963

none

964

Returns

965

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

966

Example

967

```

968     #include "papi.h"
969
970     papi_status_t status;
971     papi_service_t handle = NULL;
972     char* user_name = NULL;
973     ...
974     user_name = papiServiceGetUsername(handle);
975     if (user_name != NULL)
976     {
977         /* use the returned name */
978         ...
979     }
980     ...
981     papiServiceDestroy(handle);
982

```

983

See Also

papiServiceCreate, papiServiceSetUsername

4.10. papiServiceGetPassword

Description

Get the user password associated with the print service handle.

Syntax

```

991     char* papiServiceGetPassword(
992         papi_service_t handle );
993

```

994

Inputs

997 handle

Handle to the print service.

999

Outputs

none

Returns

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

Example

```

1006     #include "papi.h"
1007
1008     papi_status_t status;
1009     papi_service_t handle = NULL;
1010     char* password = NULL;
1011     ...
1012     password = papiServiceGetPassword(handle);
1013     if (password != NULL)
1014     {
1015         /* use the returned password */
1016         ...
1017     }
1018     ...
1019     papiServiceDestroy(handle);
1020

```


1021

1022

See Also

1023

papiServiceCreate, papiServiceSetPassword

1024 **4.11. papiServiceGetEncryption**

1025

Description

1026

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

1027

Syntax

1028

1029

1030

1031

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

1032

1033

Inputs

1034

1035

handle

1036

Handle to the print service.

1037

1038

Outputs

1039

none

1040

Returns

1041

The type of encryption associated with the print service handle.

1042

Example

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

```
#include "papi.h"

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

1055

1056

See Also

1057

papiServiceCreate, papiServiceSetEncryption

1058 **4.12. papiServiceGetAppData**

1059

Description

1060

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

1061

1062 **Syntax**

1063

```
1064           void* papiServiceGetAppData (
1065                     papi_service_t handle );
1066
```

1067

1068 **Inputs**

1069

1070 handle

 Handle to the print service.

1072

1073 **Outputs**

1074 none

1075 **Returns**

1076 A pointer to the application-specific data associated with the print service handle.

1077 **Example**

1078

```
1079           #include "papi.h"
1080
1081           papi_status_t status;
1082           papi_service_t handle = NULL;
1083           char* app_data = NULL;
1084           ...
1085           app_data = (char*)papiServiceGetAppData(handle);
1086           if (app_data != NULL)
1087           {
1088                /* use the returned application data */
1089                ...
1090           }
1091           ...
1092           papiServiceDestroy(handle);
1093
```

1094

1095 **See Also**

1096 papiServiceCreate, papiServiceSetAppData

1097 **4.13. papiServiceGetStatusMessage**

1098 **Description**

1099 Get the message associated with the status of the last operation performed. The
1100 status message returned from this function may be more detailed than the status
1101 message returned from papiStatusString (if the print service supports returning
1102 more detailed error messages).

1103 The returned message will be localized in the language of the submitter of the
1104 original operation.

1105 **Syntax**

1106

```
1107           const char* papiServiceGetStatusMessage (
```

```
1108         const papi_service_t handle );
1109
```

1110

1111 **Inputs**

1112

1113 handle

Handle to the print service.

1115

1116 **Outputs**

1117 none

1118 **Returns**

1119 Pointer to the message associated with the status of the last operation performed.

1120 **Example**

1121

```
1122 #include "papi.h"
1123
1124 papi_status_t status;
1125 papi_service_t handle = NULL;
1126 const char* user_name = "pappy";
1127 ...
1128 status = papiServiceCreate(&handle,
1129                             NULL,
1130                             NULL,
1131                             NULL,
1132                             NULL,
1133                             PAPI_ENCRYPT_IF_REQUESTED,
1134                             NULL);
1135
1136 if (status != PAPI_OK)
1137 {
1138     /* handle the error */
1139     ...
1140 }
1141
1142 status = papiServiceSetUsername(handle, user_name);
1143 if (status != PAPI_OK)
1144 {
1145     /* handle the error */
1146     fprintf(stderr, "papiServiceSetUsername failed: %s\n",
1147             papiServiceGetStatusMessage(handle));
1148     ...
1149 }
1150 ...
1151 papiServiceDestroy(handle);
```

1152

1153 **See Also**

1154 papiStatusString

1155 Chapter 5. Printer API

1156 5.1. Usage

1157 The papiPrinterQuery function queries all/some of the attributes of a printer
1158 object. It returns a list of printer attributes. A successful call to papiPrinterQuery is
1159 typically followed by code which examines and processes the returned attributes.
1160 The using program would then call papiPrinterFree to delete the returned results.

1161 Printers can be found via calls to papiPrintersList. A successful call to
1162 papiPrintersList is typically followed by code to iterate through the list of returned
1163 printers, possibly querying each (papiPrinterQuery) for further information (e.g. to
1164 restrict what printers get displayed for a particular user/request). The using
1165 program would then call papiPrinterListFree to free the returned results.

1166 5.2. papiPrintersList

1167 Description

1168 List all printers known by the print service which match the specified filter.

1169 Depending on the functionality of the target service's "printer directory", the
1170 returned list may be limited to only printers managed by a particular server or it
1171 may include printers managed by other servers.

1172 Syntax

```
1174 papi_status_t papiPrintersList(  
1175             papi_service_t   handle,  
1176             const char*      requested_attrs[],  
1177             const papi_filter_t* filter,  
1178             papi_printer_t** printers );  
1179
```

1181 Inputs

1183 handle

1184 Handle to the print service to use.

1185 requested_attrs

1186 (optional) NULL terminated array of attributes to be queried. If NULL is
1187 passed then all attributes are queried. (NOTE: The printer may return more
1188 attributes than you requested. This is merely an advisory request that may
1189 reduce the amount of data returned if the printer/server supports it.)

1190 filter

1191 (optional) Pointer to a filter to limit the number of printers returned on the list
1192 request. See Section 3.9 for details. If NULL is passed then all known printers
1193 are listed.

1195 **Outputs**

1196

1197 printers

1198 List of printer objects that matched the filter criteria.

1199

1200 **Returns**

1201 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1202 value is returned.

1203 **Example**

1204

```

1205 #include "papi.h"
1206
1207 int i;
1208 papi_status_t status;
1209 papi_service_t handle = NULL;
1210 const char* service_name = "ipp://printserv:631";
1211 const char* user_name = "pappy";
1212 const char* password = "goober";
1213 const char* req_attrs[] =
1214 {
1215     "printer-name",
1216     "printer-location",
1217     NULL
1218 };
1219
1220 const papi_filter_t filter =
1221     PAPI_PRINTER_BW | PAPI_PRINTER_DUPLEX;
1222 papi_printer_t* printers = NULL;
1223 ...
1224 status = papiServiceCreate(&handle,
1225     service_name,
1226     user_name,
1227     password,
1228     NULL,
1229     PAPI_ENCRYPT_IF_REQUESTED,
1230     NULL);
1231
1232 if (status != PAPI_OK)
1233 {
1234     /* handle the error */
1235     ...
1236 }
1237
1238 status = papiPrinterList(handle,
1239     req_attrs,
1240     filter,
1241     &printers);
1242
1243 if (status != PAPI_OK)
1244 {
1245     /* handle the error */
1246     fprintf(stderr, "papiPrinterList failed: %s\n",
1247         papiServiceGetStatusMessage(handle));
1248     ...
1249 }
1250
1251 if (printers != NULL)
1252 {
1253     for (i=0; printers[i] != NULL; i++)
1254     {
1255         /* process the printer object */
1256         ...
1257     }
1258     papiPrinterListFree(printers);
1259 }
1260
1261 papiServiceDestroy(handle);

```

1260

1261 **See Also**

1262 papiPrinterListFree, papiPrinterQuery

1263 **5.3. papiPrinterQuery**1264 **Description**

1265 Queries some or all the attributes of the specified printer object. This includes
 1266 attributes representing the capabilities of the printer, which the caller may use to
 1267 determine which print options to present to the user. How the attributes are
 1268 obtained (e.g. from a static database, from a dialog with the hardware, from a dialog
 1269 with a driver, etc.) is up to the implementer of the API and is beyond the scope of
 1270 this standard.

1271 This optionally includes "context" information which specifies job attributes in the
 1272 context of which the capabilities information is to be constructed.

1273 **Syntax**

1274

```

1275 papi_status_t papiPrinterQuery(
1276             papi_service_t   handle,
1277             const char*      name,
1278             const char*      requested_attrs[],
1279             const papi_attribute_t** job_attrs,
1280             papi_printer_t*  printer );
1281

```

1282

1283 **Inputs**

1284

1285 handle

1286 Handle to the print service to use.

1287 name

1288 The name or URI of the printer to query.

1289 requested_attrs

1290 (optional) NULL terminated array of attributes to be queried. If NULL is
 1291 passed then all attributes are queried. (NOTE: The printer may return more
 1292 attributes than you requested. This is merely an advisory request that may
 1293 reduce the amount of data returned if the printer/server supports it.)

1294 job_attrs

1295 (optional) NULL terminated array of job attributes in the context of which the
 1296 capabilities information is to be constructed. In other words, the returned
 1297 printer attributes represent the capabilities of the printer given that these
 1298 specified job attributes are requested. This allows for more accurate
 1299 information to be retrieved by the caller for a specific job (e.g. "if the job is
 1300 printed on A4 size media then duplex output is not available"). If NULL is
 1301 passed then the full capabilities of the printer are queried.

1302 Support for this argument is optional. If the underlying print system does not
 1303 have access to capabilities information bound by job context, then this
 1304 argument may be ignored. But if the calling application will be using the
 1305 returned information to build print job data, then it is always advisable to
 1306 specify the job context attributes. The more context information provided, the

1307 more accurate capabilities information is likely to be returned from the print
1308 system.

1309

1310 **Outputs**

1311

1312 printer

Pointer to a printer object containing the requested attributes.

1314

1315 **Returns**

1316 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1317 value is returned.

1318 **Example**

1319

```

1320 #include "papi.h"
1321
1322 papi_status_t status;
1323 papi_service_t handle = NULL;
1324 const char* service_name = "ipp://printserv:631";
1325 const char* user_name = "pappy";
1326 const char* password = "goober";
1327 const char* printer_name = "my-printer";
1328 const char* req_attrs[] =
1329 {
1330     "printer-name",
1331     "printer-location",
1332     "printer-state",
1333     "printer-state-reasons",
1334     "printer-state-message",
1335     NULL
1336 };
1337 papi_attribute_t** job_attrs = NULL;
1338 papi_printer_t printer = NULL;
1339 ...
1340 status = papiServiceCreate(&handle,
1341                             service_name,
1342                             user_name,
1343                             password,
1344                             NULL,
1345                             PAPI_ENCRYPT_IF_REQUESTED,
1346                             NULL);
1347
1348 if (status != PAPI_OK)
1349 {
1350     /* handle the error */
1351     ...
1352 }
1353
1354 papiAttributeListAddString(&job_attrs,
1355                             PAPI_EXCL,
1356                             "media",
1357                             "legal");
1358
1359 status = papiPrinterQuery(handle,
1360                             printer_name,
1361                             req_attrs,
1362                             job_attrs,
1363                             &printer);
1364
1365 if (status != PAPI_OK)
1366 {
1367     /* handle the error */
1368     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1369             papiServiceGetStatusMessage(handle));
1370     ...
1371 }
1372
1373 if (printer != NULL)
1374 {
1375     /* process the printer object */
1376     ...
1377     papiPrinterFree(printer);
1378 }

```

```
1378     papiAttributeListFree(job_attrs);  
1379     papiServiceDestroy(handle);  
1380
```

1381

1382 **See Also**

1383 papiPrinterList, papiPrinterFree, papiPrinterModify

1384 5.4. papiPrinterModify

1385 **Description**

1386 Modifies some or all the attributes of the specified printer object.

1387 **Syntax**

1388

```
1389     papi_status_t papiPrinterModify(  
1390         papi_service_t handle,  
1391         const char* printer_name,  
1392         const papi_attribute_t** attrs,  
1393         papi_printer_t* printer );  
1394
```

1395

1396 **Inputs**

1397

1398 handle

1399 Handle to the print service to use.

1400 printer_name

1401 Pointer to the name or URI of the printer to be modified.

1402 attrs

1403 Attributes to be modified. Any attributes not specified are left unchanged.

1404

1405 **Outputs**

1406

1407 printer

1408 The modified printer object.

1409

1410 **Returns**

1411 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1412 value is returned.

1413 **Example**

1414

```
1415     #include "papi.h"  
1416     papi_status_t status;
```



```

1418     papi_service_t handle = NULL;
1419     const char* printer_name = "my-printer";
1420     papi_printer_t printer = NULL;
1421     papi_attribute_t** attrs = NULL;
1422     ...
1423     status = papiServiceCreate(&handle,
1424                               NULL,
1425                               NULL,
1426                               NULL,
1427                               NULL,
1428                               PAPI_ENCRYPT_NEVER,
1429                               NULL);
1430
1431     if (status != PAPI_OK)
1432     {
1433         /* handle the error */
1434         ...
1435     }
1436
1437     papiAttributeListAddString(&attrs,
1438                               PAPI_EXCL,
1439                               "printer-location",
1440                               "Bldg 17/Room 234");
1441
1442     status = papiPrinterModify(handle,
1443                               printer_name,
1444                               attrs,
1445                               &printer);
1446
1447     if (status != PAPI_OK)
1448     {
1449         /* handle the error */
1450         fprintf(stderr, "papiPrinterModify failed: %s\n",
1451                papiServiceGetStatusMessage(handle));
1452         ...
1453     }
1454
1455     if (printer != NULL)
1456     {
1457         /* process the printer */
1458         ...
1459         papiPrinterFree(printer);
1460     }
1461
1462     papiServiceDestroy(handle);

```

1462

1463

See Also

1464

papiPrinterQuery, papiPrinterFree

1465

5.5. papiPrinterPause

1466

Description

1467

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

1468

1469

1470

1471

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

1472

1473

Syntax

1474

1475

```

papi_status_t papiPrinterPause(
1476     papi_service_t    handle,
1477     const char*       name,
1478     const char*       message );
1479

```

1480

1481 **Inputs**

1482

1483 handle

1484 Handle to the print service to use.

1485 name

1486 The name or URI of the printer to operate on.

1487 message

1488 (optional) An explanatory message to be associated with the paused printer.

1489 This message may be ignored if the underlying print system does not support

1490 associating a message with a paused printer.

1491

1492 **Outputs**

1493 none

1494 **Returns**

1495 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure

1496 value is returned.

1497 **Example**

1498

```

1499           #include "papi.h"
1500
1501           papi_status_t status;
1502           papi_service_t handle = NULL;
1503           const char* service_name = "ipp://printserv:631";
1504           const char* user_name = "pappy";
1505           const char* password = "goober";
1506           const char* printer_name = "my-printer";
1507           ...
1508           status = papiServiceCreate(&handle,
1509                                    service_name,
1510                                    user_name,
1511                                    password,
1512                                    NULL,
1513                                    PAPI_ENCRYPT_IF_REQUESTED,
1514                                    NULL);
1515
1516           if (status != PAPI_OK)
1517           {
1518               /* handle the error */
1519               ...
1520           }
1521
1522           status = papiPrinterPause(handle, printer_name, NULL);
1523           if (status != PAPI_OK)
1524           {
1525               /* handle the error */
1526               fprintf(stderr, "papiPrinterPause failed: %s\n",
1527                       papiServiceGetStatusMessage(handle));
1528               ...
1529           }
1530           ...
1531           papiServiceDestroy(handle);

```

1532

1533 **See Also**

1534 papiPrinterResume

1535 **5.6. papiPrinterResume**1536 **Description**

1537 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the
 1538 effects of papiPrinterPause). This operation is optional and may not be supported
 1539 by all printers/servers, but it must be supported if papiPrinterPause is supported.

1540 **Syntax**

1541

```
1542 papi_status_t papiPrinterResume(
1543             papi_service_t   handle,
1544             const char*      name );
1545
```

1546

1547 **Inputs**

1548

1549 handle

Handle to the print service to use.

1551 name

The name or URI of the printer to operate on.

1553

1554 **Outputs**

1555 none

1556 **Returns**

1557 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1558 value is returned.

1559 **Example**

1560

```
1561 #include "papi.h"
1562
1563 papi_status_t status;
1564 papi_service_t handle = NULL;
1565 const char* service_name = "ipp://printserv:631";
1566 const char* user_name = "pappy";
1567 const char* password = "goober";
1568 const char* printer_name = "my-printer";
1569 ...
1570 status = papiServiceCreate(&handle,
1571                          service_name,
1572                          user_name,
1573                          password,
1574                          NULL,
1575                          PAPI_ENCRYPT_IF_REQUESTED,
1576                          NULL);
1577
1578 if (status != PAPI_OK)
1579 {
1580     /* handle the error */
1581     ...
1582 }
1583
1584 status = papiPrinterPause(handle, printer_name);
1585 if (status != PAPI_OK)
1586 {
1587     /* handle the error */
1588     fprintf(stderr, "papiPrinterPause failed: %s\n",
1589            papiServiceGetStatusMessage(handle));
```

```

1589     ...
1590 }
1591 ...
1592 status = papiPrinterResume(handle, printer_name);
1593 if (status != PAPI_OK)
1594 {
1595     /* handle the error */
1596     fprintf(stderr, "papiPrinterResume failed: %s\n",
1597            papiServiceGetStatusMessage(handle));
1598     ...
1599 }
1600
1601 papiServiceDestroy(handle);
1602

```

1603

1604 **See Also**

1605 papiPrinterPause

1606 **5.7. papiPrinterPurgeJobs**1607 **Description**

1608 Remove all jobs from the specified printer object regardless of their states. This
 1609 includes removing jobs that have completed and are being kept for history (if any).
 1610 This operation is optional and may not be supported by all printers/servers.

1611 **Syntax**

1612

```

1613 papi_status_t papiPrinterPurgeJobs(
1614     papi_service_t    handle,
1615     const char*       name,
1616     papi_job_t**      result);
1617

```

1618

1619 **Inputs**

1620

1621 handle

1622 Handle to the print service to use.

1623 name

1624 The name or URI of the printer to operate on.

1625

1626 **Outputs**

1627

1628 result

1629 (optional) Pointer to a list of purged jobs with the identifying information (job-
 1630 id/job-uri), success/fail, and possibly a detailed message. If NULL is passed
 1631 then no job list is returned. Support for the returned job list is optional and may
 1632 not be supported by all implementations (if not supported, the function
 1633 completes with PAPI_OK_SUBST but no list is returned).

1634 name

1635 The name or URI of the printer to operate on.

1636

1637 **Returns**1638 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1639 value is returned.1640 **Example**

1641

```

1642 #include "papi.h"
1643
1644 papi_status_t status;
1645 papi_service_t handle = NULL;
1646 const char* service_name = "ipp://printserv:631";
1647 const char* user_name = "pappy";
1648 const char* password = "goober";
1649 const char* printer_name = "my-printer";
1650 ...
1651 status = papiServiceCreate(&handle,
1652                             service_name,
1653                             user_name,
1654                             password,
1655                             NULL,
1656                             PAPI_ENCRYPT_IF_REQUESTED,
1657                             NULL);
1658
1659 if (status != PAPI_OK)
1660 {
1661     /* handle the error */
1662     ...
1663 }
1664
1665 status = papiPrinterPurgeJobs(handle, printer_name);
1666 if (status != PAPI_OK)
1667 {
1668     /* handle the error */
1669     fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
1670             papiServiceGetStatusMessage(handle));
1671     ...
1672 }
1673
1674 papiServiceDestroy(handle);

```

1675

1676 **See Also**

1677 papiJobCancel

1678 **5.8. papiPrinterListJobs**1679 **Description**

1680 List print job(s) associated with the specified printer.

1681 **Syntax**

1682

```

1683 papi_status_t papiPrinterListJobs(
1684     papi_service_t handle,
1685     const char* printer,
1686     const char* requested_attrs[],
1687     const int type_mask,
1688     const int max_num_jobs,
1689     papi_job_t** jobs );
1690

```

1691

1692 **Inputs**

1693

1694 handle

1695 Handle to the print service to use.

1696 requested_attrs

1697 (optional) NULL terminated array of attributes to be queried. If NULL is
 1698 passed then all available attributes are queried. (NOTE: The printer may return
 1699 more attributes than you requested. This is merely an advisory request that
 1700 may reduce the amount of data returned if the printer/server supports it.)

1701 type_mask

1702 A bit mask which determines what jobs will get returned. The following
 1703 constants can be bitwise-OR-ed together to select which types of jobs to list:

```

1704           #define PAPI_LIST_JOBS_OTHERS        0x0001 /* return jobs other than
1705                                                    those submitted by the
1706                                                    user name assoc with
1707                                                    the handle */
1708           #define PAPI_LIST_JOBS_COMPLETED     0x0002 /* return completed jobs */
1709           #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
1710                                                    jobs */
1711           #define PAPI_LIST_JOBS_ALL           0xFFFF /* return all jobs */

```

1713

1714 max_num_jobs

1715 Limit to the number of jobs returned. If 0 is passed, then there is no limit on
 1716 the number of jobs which may be returned.

1717

1718 **Outputs**

1719

1720 jobs

1721 List of job objects returned.

1722

1723 **Returns**

1724 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1725 value is returned.

1726 **Example**

1727

```

1728           #include "papi.h"
1729
1730           int i;
1731           papi_status_t status;
1732           papi_service_t handle = NULL;
1733           const char* printer_name = "my-printer";
1734           papi_job_t* jobs = NULL;
1735           const char* job_attrs[] =
1736           {
1737            "job-id",
1738            "job-name",
1739            "job-originating-user-name",
1740            "job-state",
1741            "job-state-reasons",

```

```

1742     NULL
1743 };
1744 ...
1745 status = papiServiceCreate(&handle,
1746     NULL,
1747     NULL,
1748     NULL,
1749     NULL,
1750     PAPI_ENCRYPT_NEVER,
1751     NULL);
1752
1753 if (status != PAPI_OK)
1754 {
1755     /* handle the error */
1756     ...
1757 }
1758
1759 status = papiPrinterListJobs(handle,
1760     printer_name,
1761     job_attrs,
1762     PAPI_LIST_JOBS_ALL,
1763     0,
1764     &jobs);
1765
1766 if (status != PAPI_OK)
1767 {
1768     /* handle the error */
1769     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
1770         papiServiceGetStatusMessage(handle));
1771     ...
1772 }
1773
1774 if (jobs != NULL)
1775 {
1776     for(i=0; jobs[i] != NULL; i++)
1777     {
1778         /* process the job */
1779         ...
1780     }
1781     papiJobListFree(jobs);
1782 }
1783
1784 papiServiceDestroy(handle);

```

1784

1785 **See Also**

1786 papiJobQuery, papiJobListFree

1787 **5.9. papiPrinterGetAttributeList**1788 **Description**

1789 Get the attribute list associated with a printer object.

1790 **Syntax**

1791

```

1792 papi_attribute_t** papiPrinterGetAttributeList(
1793     papi_printer_t printer );
1794

```

1795

1796 **Inputs**

1797

1798 printer

1799 Handle of the printer object.

1800

1801 **Outputs**

1802 none

1803

Returns

1804

Pointer to the attribute list associated with the printer object.

1805

Example

1806

```

1807 #include "papi.h"
1808
1809 papi_status_t status;
1810 papi_service_t handle = NULL;
1811 const char* printer_name = "my-printer";
1812 papi_printer_t printer = NULL;
1813 papi_attribute_list* attrs = NULL;
1814 ...
1815 status = papiServiceCreate(&handle,
1816                             NULL,
1817                             NULL,
1818                             NULL,
1819                             NULL,
1820                             PAPI_ENCRYPT_NEVER,
1821                             NULL);
1822
1823 if (status != PAPI_OK)
1824 {
1825     /* handle the error */
1826     ...
1827 }
1828
1829 status = papiPrinterQuery(handle,
1830                           printer_name,
1831                           NULL,
1832                           &printer);
1833
1834 if (status != PAPI_OK)
1835 {
1836     /* handle the error */
1837     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1838             papiServiceGetStatusMessage(handle));
1839     ...
1840 }
1841
1842 if (printer != NULL)
1843 {
1844     /* process the printer object */
1845     attrs = papiPrinterGetAttributeList(printer);
1846     ...
1847     papiPrinterFree(printer);
1848 }
1849
1850 papiServiceDestroy(handle);

```

1850

1851

See Also

1852

papiPrintersList, papiPrinterQuery

1853

5.10. papiPrinterFree

1854

Description

1855

Free a printer object.

1856

Syntax

1857

1858

```

1859 void papiPrinterFree(
1860         papi_printer_t printer );

```

1861

1862

Inputs

1863

1864 printer

1865 Handle of the printer object to free.

1866

1867 **Outputs**

1868 none

1869 **Returns**

1870 none

1871 **Example**

1872

```

1873 #include "papi.h"
1874
1875 papi_status_t status;
1876 papi_service_t handle = NULL;
1877 const char* printer_name = "my-printer";
1878 papi_printer_t printer = NULL;
1879 ...
1880 status = papiServiceCreate(&handle,
1881                            NULL,
1882                            NULL,
1883                            NULL,
1884                            NULL,
1885                            PAPI_ENCRYPT_NEVER,
1886                            NULL);
1887
1888 if (status != PAPI_OK)
1889 {
1890     /* handle the error */
1891     ...
1892 }
1893
1894 status = papiPrinterQuery(handle,
1895                           printer_name,
1896                           NULL,
1897                           &printer);
1898
1899 if (status != PAPI_OK)
1900 {
1901     /* handle the error */
1902     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1903            papiServiceGetStatusMessage(handle));
1904     ...
1905 }
1906
1907 if (printer != NULL)
1908 {
1909     /* process the printer object */
1910     ...
1911     papiPrinterFree(printer);
1912 }
1913
1914 papiServiceDestroy(handle);

```

1914

1915 **See Also**

1916 papiPrinterQuery

1917 5.11. papiPrinterListFree

1918 **Description**

1919 Free a list of printer objects.

1920 **Syntax**

1921

```

1922 void papiPrinterListFree (
1923     papi_printer_t*   printers );

```

1924

1925

1926 **Inputs**

1927

1928 printers

1929 Pointer to the printer object list to free.

1930

1931 **Outputs**

1932 none

1933 **Returns**

1934 none

1935 **Example**

1936

```

1937 #include "papi.h"
1938
1939 papi_status_t status;
1940 papi_service_t handle = NULL;
1941 const char* printer_name = "my-printer";
1942 papi_printer_t* printers = NULL;
1943 ...
1944 status = papiServiceCreate(&handle,
1945                          NULL,
1946                          NULL,
1947                          NULL,
1948                          NULL,
1949                          PAPI_ENCRYPT_NEVER,
1950                          NULL);
1951
1952 if (status != PAPI_OK)
1953 {
1954     /* handle the error */
1955     ...
1956 }
1957
1958 status = papiPrinterList(handle,
1959                          NULL,
1960                          NULL,
1961                          &printers);
1962
1963 if (status != PAPI_OK)
1964 {
1965     /* handle the error */
1966     fprintf(stderr, "papiPrinterList failed: %s\n",
1967            papiServiceGetStatusMessage(handle));
1968     ...
1969 }
1970
1971 if (printers != NULL)
1972 {
1973     /* process the printer objects */
1974     ...
1975     papiPrinterListFree(printers);
1976 }
1977
1978 papiServiceDestroy(handle);

```

1978

1979 **See Also**

1980 papiPrinterList

1981 Chapter 6. Attributes API

1982 6.1. papiAttributeListAdd

1983 Description

1984 Add an attribute/value to an attribute list. Depending on the `add_flags`, this may
1985 also be used to add values to an existing multivalued attribute. Memory is allocated
1986 and copies of the input arguments are created. It is the caller's responsibility to call
1987 `papiAttributeListFree` when done with the attribute list.

1988 This function is equivalent to the `papiAttributeListAddString`,
1989 `papiAttributeListAddInteger`, etc. functions defined later in this chapter.

1990 Syntax

1991

```
1992 papi_status_t papiAttributeListAdd(  
1993     papi_attribute_t*** attrs,  
1994     const int add_flags,  
1995     const char* name,  
1996     const papi_attribute_value_type_t type,  
1997     const papi_attribute_value_t* value );  
1998
```

1999

2000 Inputs

2001

2002 `attrs`

2003 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
2004 `NULL` then this function will allocate the attribute list.

2005 `add_flags`

2006 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
2007 that indicates how to handle the request.

2008 `name`

2009 Points to the name of the attribute to add.

2010 `type`

2011 The type of values for this attribute.

2012 `value`

2013 Points to the attribute value to be added.

2014

2015 Outputs

2016

2017 `attrs`

2018 The attribute list is updated.

2019

2020

Returns

2021

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

2022

2023

Example

2024

2025

```
#include "papi.h"
papi_attribute_t** attrs = NULL;
...
papiAttributeListAdd(&attrs,
                    PAPI_EXCL,
                    "job-name",
                    PAPI_STRING,
                    "My job" );
...
papiAttributeListFree(attrs);
```

2026

2027

2028

2029

2030

2031

2032

2033

2034

2035

2036

2037

2038

See Also

2039

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

2040

2041

2042

6.2. papiAttributeListAddString

2043

Description

2044

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

2045

2046

2047

2048

Syntax

2049

2050

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

2051

2052

2053

2054

2055

2056

2057

Inputs

2058

2059

attrs

2060

Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is `NULL` then this function will allocate the attribute list.

2061

2062

add_flags

2063

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

2064

2065 name
 2066 Points to the name of the attribute to add.

2067 value
 2068 The value to be added.

2069

2070 **Outputs**

2071

2072 attrs
 2073 The attribute list is updated.

2074

2075 **Returns**

2076 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2077 value is returned.

2078 **Example**

2079

```
2080 #include "papi.h"
2081
2082 papi_attribute_t** attrs = NULL;
2083 ...
2084 papiAttributeListAddString(&attrs,
2085                             PAPI_EXCL,
2086                             "job-name",
2087                             "My job" );
2088 ...
2089 papiAttributeListFree(attrs);
2090
```

2091

2092 **See Also**

2093 papiAttributeListFree, papiAttributeListAdd

2094 **6.3. papiAttributeListAddInteger**

2095 **Description**

2096 Add an integer-valued attribute to an attribute list. Depending on the add_flags,
 2097 this may also be used to add values to an existing multivalued attribute. Memory is
 2098 allocated and copies of the input arguments are created. It is the caller's
 2099 responsibility to call papiAttributeListFree when done with the attribute list.

2100 **Syntax**

2101

```
2102 papi_status_t papiAttributeListAddInteger(
2103     papi_attribute_t*** attrs,
2104     const int add_flags,
2105     const char* name,
2106     const int value );
2107
```

2108

2109 **Inputs**

2110

2111 attrs

2112 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2113 NULL then this function will allocate the attribute list.

2114 add_flags

2115 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2116 that indicates how to handle the request.

2117 name

2118 Points to the name of the attribute to add.

2119 value

2120 The value to be added.

2121

2122 **Outputs**

2123

2124 attrs

2125 The attribute list is updated.

2126

2127 **Returns**

2128 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2129 value is returned.

2130 **Example**

2131

```

2132           #include "papi.h"
2133           papi_attribute_t** attrs = NULL;
2134           ...
2135           papiAttributeListAddInteger(&attrs,
2136                                       PAPI_EXCL,
2137                                       "copies",
2138                                       3 );
2139           ...
2140           papiAttributeListFree(attrs);
2141           ...
2142           ...

```

2143

2144 **See Also**

2145 papiAttributeListFree, papiAttributeListAdd

2146 **6.4. papiAttributeListAddBoolean**2147 **Description**

2148 Add a boolean-valued attribute to an attribute list. Depending on the add_flags,
 2149 this may also be used to add values to an existing multivalued attribute. Memory is
 2150 allocated and copies of the input arguments are created. It is the caller's
 2151 responsibility to call papiAttributeListFree when done with the attribute list.

2152 **Syntax**

2153

```

2154     papi_status_t papiAttributeListAddBoolean(
2155         papi_attribute_t*** attrs,
2156         const int add_flags,
2157         const char* name,
2158         const char value );
2159 
```

2160

2161 **Inputs**

2162

2163 attrs

2164 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2165 NULL then this function will allocate the attribute list.

2166 add_flags

2167 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2168 that indicates how to handle the request.

2169 name

2170 Points to the name of the attribute to add.

2171 value

2172 The value (PAPI_FALSE or PAPI_TRUE) to be added.

2173

2174 **Outputs**

2175

2176 attrs

2177 The attribute list is updated.

2178

2179 **Returns**

2180 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2181 value is returned.

2182 **Example**

2183

```

2184     #include "papi.h"
2185
2186     papi_attribute_t** attrs = NULL;
2187     ...
2188     papiAttributeListAddBoolean(&attrs,
2189                               PAPI_EXCL,
2190                               "color-supported",
2191                               PAPI_TRUE );
2192     ...
2193     papiAttributeListFree(attrs);
2194 
```

2195

2196 **See Also**
2197 papiAttributeListFree, papiAttributeListAdd

2198 **6.5. papiAttributeListAddRange**

2199 **Description**

2200 Add a range-valued attribute to an attribute list. Depending on the `add_flags`, this
2201 may also be used to add values to an existing multivalued attribute. Memory is
2202 allocated and copies of the input arguments are created. It is the caller's
2203 responsibility to call `papiAttributeListFree` when done with the attribute list.

2204 **Syntax**

```
2205  
2206           papi_status_t papiAttributeListAddRange(  
2207           papi_attribute_t*** attrs,  
2208           const int add_flags,  
2209           const char* name,  
2210           const int lower,  
2211           const int upper );  
2212
```

2213

2214 **Inputs**

2215

2216 attrs

2217 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
2218 `NULL` then this function will allocate the attribute list.

2219 add_flags

2220 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
2221 that indicates how to handle the request.

2222 name

2223 Points to the name of the attribute to add.

2224 lower

2225 The lower range value. This value must be less than or equal to the upper
2226 range value.

2227 upper

2228 The upper range value. This value must be greater than or equal to the lower
2229 range value.

2230

2231 **Outputs**

2232

2233 attrs

2234 The attribute list is updated.

2235

2236

Returns

2237

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

2238

2239

Example

2240

2241

```
#include "papi.h"
2242
2243 papi_attribute_t** attrs = NULL;
2244 ...
2245 papiAttributeListAddRange (&attrs,
2246                             PAPI_EXCL,
2247                             "job-k-octets-supported",
2248                             1,
2249                             100000 );
2250 ...
2251 papiAttributeListFree (attrs);
2252
```

2253

2254

See Also

2255

papiAttributeListFree

2256

6.6. papiAttributeListAddResolution

2257

Description

2258

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

2259

2260

2261

2262

Syntax

2263

2264

```
papi_status_t papiAttributeListAddResolution(
2265     papi_attribute_t*** attrs,
2266     const int add_flags,
2267     const char* name,
2268     const int xres,
2269     const int yres,
2270     const papi_res_t units );
2271
```

2272

2273

Inputs

2274

2275

attrs

2276

Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is NULL then this function will allocate the attribute list.

2277

2278

add_flags

2279

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

2280

2281 name
 2282 Points to the name of the attribute to add.

2283 xres
 2284 The X-axis resolution value.

2285 yres
 2286 The Y-axis resolution value.

2287 units
 2288 The units of the resolution values provided.

2289

2290 **Outputs**

2291

2292 attrs
 2293 The attribute list is updated.

2294

2295 **Returns**

2296 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2297 value is returned.

2298 **Example**

2299

```

2300 #include "papi.h"
2301
2302 papi_attribute_t** attrs = NULL;
2303 ...
2304 papiAttributeListAddResolution(&attrs,
2305                               PAPI_EXCL,
2306                               "printer-resolution",
2307                               300,
2308                               300,
2309                               PAPI_RES_PER_INCH );
2310 ...
2311 papiAttributeListFree(attrs);
2312
    
```

2313

2314 **See Also**

2315 papiAttributeListFree

2316 **6.7. papiAttributeListAddDatetime**

2317 **Description**

2318 Add a date/time-valued attribute to an attribute list. Depending on the add_flags,
 2319 this may also be used to add values to an existing multivalued attribute. Memory is
 2320 allocated and copies of the input arguments are created. It is the caller's
 2321 responsibility to call papiAttributeListFree when done with the attribute list.

2322 **Syntax**

2323

```

2324     papi_status_t papiAttributeListAddDatetime(
2325         papi_attribute_t*** attrs,
2326         const int add_flags,
2327         const char* name,
2328         const time_t date_time );
2329

```

2330

2331 **Inputs**

2332

2333 attrs

2334 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2335 NULL then this function will allocate the attribute list.

2336 add_flags

2337 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2338 that indicates how to handle the request.

2339 name

2340 Points to the name of the attribute to add.

2341 date_time

2342 The date/time value.

2343

2344 **Outputs**

2345

2346 attrs

2347 The attribute list is updated.

2348

2349 **Returns**

2350 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2351 value is returned.

2352 **Example**

2353

```

2354     #include "papi.h"
2355
2356     papi_attribute_t** attrs = NULL;
2357     time_t date_time
2358     ...
2359     time(&date_time);
2360     papiAttributeListAddDatetime(&attrs,
2361         PAPI_EXCL,
2362         "date-time-at-creation",
2363         date_time );
2364     ...
2365     papiAttributeListFree(attrs);
2366

```

2367

2368 **See Also**

2369 `papiAttributeListFree`

2370 **6.8. papiAttributeListAddCollection**

2371 **Description**

2372 Add a collection-valued attribute to an attribute list. Depending on the `add_flags`,
2373 this may also be used to add values to an existing multivalued attribute. Memory is
2374 allocated and copies of the input arguments are created. It is the caller's
2375 responsibility to call `papiAttributeListFree` when done with the attribute list.

2376 **Syntax**

2377

```
2378 papi_status_t papiAttributeListAddCollection(  
2379     papi_attribute_t*** attrs,  
2380     const int add_flags,  
2381     const char* name,  
2382     const papi_attribute_t** collection );  
2383
```

2384

2385 **Inputs**

2386

2387 `attrs`

2388 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
2389 `NULL` then this function will allocate the attribute list.

2390 `add_flags`

2391 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
2392 that indicates how to handle the request.

2393 `name`

2394 Points to the name of the attribute to add.

2395 `collection`

2396 The collection value.

2397

2398 **Outputs**

2399

2400 `attrs`

2401 The attribute list is updated.

2402

2403 **Returns**

2404 If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate failure
2405 value is returned.

2406

Example

2407

2408

```

#include "papi.h"

papi_attribute_t** attrs = NULL;
papi_attribute_t** collection = NULL;
...
/* Build the collection attribute */
papiAttributeListAddString(&collection,
    PAPI_EXCL,
    "media-key",
    "iso-a4-white");
papiAttributeListAddString(&collection,
    PAPI_EXCL,
    "media-type",
    "stationery");
...
/* Add the collection attribute */
papiAttributeListAddCollection(&attrs,
    PAPI_EXCL,
    "media-col",
    collection );
...
papiAttributeListFree (collection);
papiAttributeListFree (attrs);

```

2409

2410

2411

2412

2413

2414

2415

2416

2417

2418

2419

2420

2421

2422

2423

2424

2425

2426

2427

2428

2429

2430

2431

2432

2433

See Also

2434

papiAttributeListFree

2435

6.9. papiAttributeDelete

2436

Description

2437

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

2438

2439

Syntax

2440

2441

```

papi_status_t papiAttributeDelete(
    papi_attribute_t*** attrs,
    const char* name);

```

2442

2443

2444

2445

2446

Inputs

2447

2448 attrs

Points to an attribute list.

2449

2450 name

Points to the name of the attribute to delete.

2451

2452

2453

Outputs

2454

2455 attrs

2456 The attribute list is updated.

2457

2458 **Returns**2459 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2460 value is returned.2461 **Example**

2462

```

2463 #include "papi.h"
2464
2465 papi_attribute_t** attrs = NULL;
2466 ...
2467 papiAttributeDelete(&attrs,
2468                   "copies" );
2469 ...
2470

```

2471

2472 **See Also**

2473 papiAttributeListFree

2474 **6.10. papiAttributeListGetValue**2475 **Description**

2476 Get an attribute's value from an attribute list.

2477 This function is equivalent to the papiAttributeListGetString,
2478 papiAttributeListGetInteger, etc. functions defined later in this chapter.2479 **Syntax**

2480

```

2481 papi_status_t papiAttributeListGetValue(
2482     const papi_attribute_t** attrs,
2483     void** iterator,
2484     const char* name,
2485     const papi_attribute_value_type_t type,
2486     papi_attribute_value_t* value );
2487

```

2488

2489 **Inputs**

2490

2491 attrs

2492 The attribute list.

2493 iterator

2494 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
2495 then only the first value is returned, even if the attribute is multivalued. If the
2496 argument points to a void* that is set to NULL, then the first attribute value is
2497 returned and the iterator can then be passed in unchanged on subsequent calls
2498 to this function to get the remaining values.

2499 name
 2500 Points to the name of the attribute whose value to get.

2501 type
 2502 The type of values for this attribute.

2503

2504 **Outputs**

2505

2506 value
 2507 Points to the attribute value to be returned.

2508

2509 **Returns**

2510 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2511 value is returned.

2512 **Example**

2513

```

2514 #include "papi.h"
2515
2516 papi_attribute_t** attrs = NULL;
2517 char* job_name_value = NULL;
2518 ...
2519 papiAttributeListGetValue(attrs,
2520     NULL,
2521     "job-name",
2522     PAPI_STRING,
2523     &job_name_value );
2524 if (job_name_value != NULL)
2525 {
2526     /* process the value */
2527     ...
2528 }
2529 ...
2530 papiAttributeListFree(attrs);
2531

```

2532

2533 **See Also**

2534 papiAttributeListFree, papiAttributeListGetString, papiAttributeListGetInteger,
 2535 papiAttributeListGetBoolean, papiAttributeListGetRange,
 2536 papiAttributeListGetResolution, papiAttributeListGetDatetime

2537 6.11. papiAttributeListGetString

2538 **Description**

2539 Get a string-valued attribute's value from an attribute list.

2540 **Syntax**

2541

```

2542 papi_status_t papiAttributeListGetString(
2543     const papi_attribute_t** attrs,
2544     void** iterator,
2545     const char* name,
2546     char** value );

```

2547

2548

2549 **Inputs**

2550

2551 attrs

2552 The attribute list.

2553 iterator

2554 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2555 then only the first value is returned, even if the attribute is multivalued. If the
 2556 argument points to a void* that is set to NULL, then the first attribute value is
 2557 returned and the iterator can then be passed in unchanged on subsequent calls
 2558 to this function to get the remaining values.

2559 name

2560 Points to the name of the attribute whose value to get.

2561

2562 **Outputs**

2563

2564 value

2565 Pointer to the char* where a pointer to the value is returned.

2566

2567 **Returns**

2568 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2569 value is returned.

2570 **Example**

2571

```

2572 #include "papi.h"
2573
2574 papi_attribute_t** attrs = NULL;
2575 char* job_name_value = NULL;
2576 ...
2577 papiAttributeListGetString(attrs,
2578                             NULL,
2579                             "job-name",
2580                             &job_name_value );
2581 if (job_name_value != NULL)
2582 {
2583     /* process the value */
2584     ...
2585 }
2586 ...
2587 papiAttributeListFree(attrs);
2588

```

2589

2590 **See Also**

2591 papiAttributeListFree, papiAttributeListGetValue

2592 6.12. papiAttributeListGetInteger

2593 Description

2594 Get an integer-valued attribute's value from an attribute list.

2595 Syntax

2596

```
2597 papi_status_t papiAttributeListGetInteger(
2598     const papi_attribute_t** attrs,
2599     void** iterator,
2600     const char* name,
2601     int* value );
2602
```

2603

2604 Inputs

2605

2606 attrs

2607 The attribute list.

2608 iterator

2609 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2610 then only the first value is returned, even if the attribute is multivalued. If the
 2611 argument points to a void* that is set to NULL, then the first attribute value is
 2612 returned and the iterator can then be passed in unchanged on subsequent calls
 2613 to this function to get the remaining values.

2614 name

2615 Points to the name of the attribute whose value to get.

2616

2617 Outputs

2618

2619 value

2620 Pointer to the int where the value is returned.

2621

2622 Returns

2623 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2624 value is returned.

2625 Example

2626

```
2627 #include "papi.h"
2628
2629 papi_attribute_t** attrs = NULL;
2630 int copies = 0;
2631 ...
2632 papiAttributeListGetInteger(attrs,
2633     NULL,
2634     "copies",
2635     &copies );
```

```
2636     /* process the value */  
2637     ...  
2638     papiAttributeListFree(attrs);  
2639
```

2640

2641 **See Also**

2642 papiAttributeListFree, papiAttributeListGetValue

2643 **6.13. papiAttributeListGetBoolean**

2644 **Description**

2645 Get an boolean-valued attribute's value from an attribute list.

2646 **Syntax**

2647

```
2648     papi_status_t papiAttributeListGetBoolean(  
2649         const papi_attribute_t** attrs,  
2650         void** iterator,  
2651         const char* name,  
2652         char* value );  
2653
```

2654

2655 **Inputs**

2656

2657 attrs

2658 The attribute list.

2659 iterator

2660 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
2661 then only the first value is returned, even if the attribute is multivalued. If the
2662 argument points to a void* that is set to NULL, then the first attribute value is
2663 returned and the iterator can then be passed in unchanged on subsequent calls
2664 to this function to get the remaining values.

2665 name

2666 Points to the name of the attribute whose value to get.

2667

2668 **Outputs**

2669

2670 value

2671 Pointer to the char where the value is returned.

2672

2673 **Returns**

2674 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2675 value is returned.

2676

Example

2677

2678

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

2679

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

2680

```
char color_supp = PAPI_FALSE;
```

2681

```
...
```

2682

```
papiAttributeListGetBoolean(attrs,
```

2683

```
    NULL,
```

2684

```
    "color-supported",
```

2685

```
    &color_supp );
```

2686

```
/* process the value */
```

2687

```
...
```

2688

```
papiAttributeListFree(attrs);
```

2689

2690

2691

See Also

2692

papiAttributeListFree, papiAttributeListGetValue

2693

6.14. papiAttributeListGetRange

2694

Description

2695

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

2696

Syntax

2697

2698

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

2699

2700

2701

2702

2703

2704

2705

2706

2707

Inputs

2708

2709

attrs

The attribute list.

2710

2711

iterator

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

2712

2713

2714

2715

2716

2717

name

Points to the name of the attribute whose value to get.

2718

2719

2720

Outputs

2721

2722 lower
 2723 Pointer to the int where the lower range value is returned.

2724 upper
 2725 Pointer to the int where the upper range value is returned.

2726
 2727 **Returns**

2728 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2729 value is returned.

2730 **Example**

2731

```

2732 #include "papi.h"
2733
2734 papi_attribute_t** attrs = NULL;
2735 int lower = 0;
2736 int upper = 0;
2737 ...
2738 papiAttributeListGetRange(attrs,
2739                            NULL,
2740                            "job-k-octets-supported",
2741                            &lower,
2742                            &upper );
2743 /* process the value */
2744 ...
2745 papiAttributeListFree(attrs);
2746

```

2747

2748 **See Also**

2749 papiAttributeListFree, papiAttributeListGetValue

2750 **6.15. papiAttributeListGetResolution**

2751 **Description**

2752 Get a resolution-valued attribute's value from an attribute list.

2753 **Syntax**

2754

```

2755 papi_status_t papiAttributeListGetResolution(
2756                   const papi_attribute_t** attrs,
2757                   void** iterator,
2758                   const char* name,
2759                   int* xres,
2760                   int* yres,
2761                   papi_res_t* units );
2762

```

2763

2764 **Inputs**

2765

2766 attrs

2767 The attribute list.

2768 iterator
 2769 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2770 then only the first value is returned, even if the attribute is multivalued. If the
 2771 argument points to a void* that is set to NULL, then the first attribute value is
 2772 returned and the iterator can then be passed in unchanged on subsequent calls
 2773 to this function to get the remaining values.

2774 name
 2775 Points to the name of the attribute whose value to get.

2776

2777 **Outputs**

2778

2779 xres
 2780 Pointer to the int where the X-resolution value is returned.

2781 yres
 2782 Pointer to the int where the Y-resolution value is returned.

2783 units
 2784 Pointer to the variable where the resolution-units value is returned.

2785

2786 **Returns**

2787 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2788 value is returned.

2789 **Example**

2790

```

2791 #include "papi.h"
2792
2793 papi_attribute_t** attrs = NULL;
2794 int xres = 0;
2795 int yres = 0;
2796 papi_res_t units;
2797 ...
2798 papiAttributeListGetResolution(attrs,
2799     NULL,
2800     "printer-resolution",
2801     &xres,
2802     &yres,
2803     &units );
2804 /* process the value */
2805 ...
2806 papiAttributeListFree(attrs);
2807

```

2808

2809 **See Also**

2810 papiAttributeListFree, papiAttributeListGetValue

2811 **6.16. papiAttributeListGetDatetime**

2812 **Description**

2813 Get a date/time-valued attribute's value from an attribute list.

2814 **Syntax**

2815

```

2816     papi_status_t papiAttributeListGetDatetime(
2817         const papi_attribute_t** attrs,
2818         void** iterator,
2819         const char* name,
2820         time_t* date_time );
2821

```

2822

2823 **Inputs**

2824

2825 attrs

2826 The attribute list.

2827 iterator

2828 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only the first value is returned, even if the attribute is multivalued. If the argument points to a void* that is set to NULL, then the first attribute value is returned and the iterator can then be passed in unchanged on subsequent calls to this function to get the remaining values.

2833 name

2834 Points to the name of the attribute whose value to get.

2835

2836 **Outputs**

2837

2838 date_time

2839 Pointer to the variable where the date/time value is returned.

2840

2841 **Returns**

2842 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

2844 **Example**

2845

```

2846     #include "papi.h"
2847
2848     papi_attribute_t** attrs = NULL;
2849     time_t date_time;
2850     ...
2851     papiAttributeListGetDatetime(attrs,
2852         NULL,
2853         "date-time-at-creation",
2854         &date_time );
2855     /* process the value */
2856     ...
2857     papiAttributeListFree(attrs);
2858

```

2859

2860

See Also

2861

papiAttributeListFree, papiAttributeListGetValue

2862 **6.17. papiAttributeListGetCollection**

2863

Description

2864

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

2865

Syntax

2866

2867

```
papi_status_t papiAttributeListGetCollection(
2868     const papi_attribute_t** attrs,
2869     void** iterator,
2870     const char* name,
2871     papi_attribute_t*** collection );
2872
```

2873

2874

Inputs

2875

2876 attrs

The attribute list.

2878 iterator

2879

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

2880

2881

2882

2883

2884 name

2885

Points to the name of the attribute whose value to get.

2886

2887

Outputs

2888

2889 collection

2890

Pointer to the attribute list where a pointer to the collection value is returned. Note that the value is not copied, so the caller does not need to free the returned list (it will be freed when the containing attribute list is freed).

2891

2892

2893

2894

Returns

2895

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

2896

2897

Example

2898

```

2899     #include "papi.h"
2900
2901     papi_attribute_t** attrs = NULL;
2902     papi_attribute_t** collection = NULL;
2903     time_t date_time;
2904     ...
2905     papiAttributeListGetCollection(attrs,
2906                                   NULL,
2907                                   "media-col",
2908                                   &collection );
2909     /* process the value */
2910     ...
2911     papiAttributeListFree(attrs);
2912

```

2913

2914 **See Also**

2915 papiAttributeListFree, papiAttributeListGetValue

2916 6.18. papiAttributeListFree

2917 Description

2918 Frees an attribute list.

2919 Syntax

2920

```

2921 void papiAttributeListFree(
2922     const papi_attribute_t** attrs );
2923

```

2924

2925 Inputs

2926

2927 attrs

2928 Attribute list to be freed.

2929

2930 Outputs

2931 none

2932 Returns

2933 none

2934 Example

2935

```

2936     #include "papi.h"
2937
2938     papi_attribute_t** attrs = NULL;
2939     ...
2940     papiAttributeListAddString(&attrs,
2941                               "job-name",
2942                               PAPI_EXCL,
2943                               1,
2944                               "My job" );
2945     ...
2946     papiAttributeListFree(attrs);
2947

```

2948

2949 **See Also**
 2950 papiAttributeListAddString, etc.

2951 **6.19. papiAttributeListFind**

2952 **Description**

2953 Find an attribute in an attribute list.

2954 **Syntax**

2955

```
2956           papi_attribute_t* papiAttributeListFind(
2957                            const papi_attribute_t** attrs,
2958                            const char*                name );
2959
```

2960

2961 **Inputs**

2962

2963 attrs
 2964 Attribute list to be searched.

2965 name
 2966 Pointer to the name of the attribute to find.

2967

2968 **Outputs**

2969 none

2970 **Returns**

2971 Pointer to the found attribute. NULL indicates that the specified attribute was not
 2972 found

2973 **Example**

2974

```
2975           #include "papi.h"
2976
2977           papi_attribute_t** attrs = NULL;
2978           papi_attribute_t* attr = NULL;
2979           ...
2980           attr = papiAttributeListFind(&attrs,
2981                                        "job-name" );
2982           if (attr != NULL)
2983           {
2984               /* process the attribute */
2985               ...
2986           }
2987           ...
2988           papiAttributeListFree(attrs);
2989
```

2990

2991 **See Also**

2992 papiAttributeListGetNext

2993 **6.20. papiAttributeListGetNext**2994 **Description**

2995 Get the next attribute in an attribute list.

2996 **Syntax**

2997

```

2998 papi_attribute_t* papiAttributeListGetNext(
2999     const papi_attribute_t** attrs,
3000     void** iterator );
3001

```

3002

3003 **Inputs**

3004

3005 attrs

3006 Attribute list to be used.

3007 iterator

3008 Pointer to an opaque (void*) iterator. This should be NULL to find the first
 3009 attribute and then passed in unchanged on subsequent calls to this function.

3010

3011 **Outputs**

3012 none

3013 **Returns**

3014 Pointer to the found attribute. NULL indicates that the end of the attribute list was
 3015 reached.

3016 **Example**

3017

```

3018 #include "papi.h"
3019
3020 papi_attribute_t** attrs = NULL;
3021 papi_attribute_t* attr = NULL;
3022 void* iterator = NULL;
3023 ...
3024 attr = papiAttributeListGetNext(&attrs,
3025                                &iterator );
3026
3027 while (attr != NULL)
3028 {
3029     /* process this attribute */
3030     ...
3031     attr = papiAttributeListGetNext(&attrs,
3032                                     &iterator );
3033 }
3034 ...
3035 papiAttributeListFree(attrs);

```

3036

3037 **See Also**

3038 papiAttributeListFind

3039 6.21. papiAttributeListFromString

3040 **Description**

3041 Convert a string of text options to an attribute list.

3042 PAPI provides two functions which map job attributes to and from text options
 3043 that are typically provided on the command-line by the user. This text encoding is
 3044 also backwards-compatible with existing printing systems and is relatively simple
 3045 to parse and generate. See Appendix A for a definition of the string syntax.

3046 **Syntax**

3047

```
3048 papi_status_t papiAttributeListFromString(  
3049     papi_attribute_t*** attrs,  
3050     const int add_flags,  
3051     const char* buffer );  
3052
```

3053

3054 **Inputs**

3055

3056 `attrs`

3057 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
 3058 `NULL` then this function will allocate the attribute list.

3059 `add_flags`

3060 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 3061 that indicates how to handle the request.

3062 `buffer`

3063 Points to text options.

3064

3065 **Outputs**

3066

3067 `attrs`

3068 The attribute list is updated.

3069

3070 **Returns**

3071 If the text string is successfully converted to an attribute list, a value of `PAPI_OK` is
 3072 returned. Otherwise an appropriate failure value is returned.

3073 **Example**

3074

```
3075 #include "papi.h"  
3076  
3077 papi_attribute_t** attrs = NULL;  
3078 char buffer[8192];  
3079 ...  
3080 strcpy(buffer,
```

```

3081     "copies=1 job-name=John's\ Really\040Nice\ Job");
3082
3083     papiAttributeListFromString(&attrs, PAPI_EXCL, buffer);
3084     ...
3085     papiAttributeListFree(attrs);
3086

```

3087

3088 **See Also**3089 `papiAttributeListToString`3090 **6.22. papiAttributeListToString**3091 **Description**

3092 Convert an attribute list to its text representation. The destination string is limited
 3093 to at most (buflen - 1) bytes plus the trailing nul byte.

3094 PAPI provides two functions which map job attributes to and from text options
 3095 that are typically provided on the command-line by the user. This text encoding is
 3096 also backwards-compatible with existing printing systems and is relatively simple
 3097 to parse and generate. See Appendix A for a definition of the string syntax.

3098 **Syntax**

3099

```

3100 papi_status_t papiAttributeListToString(
3101     const papi_attribute_t** attrs,
3102     char* buffer,
3103     const int buflen );
3104

```

3105

3106 **Inputs**

3107

3108 `attrs`

3109 Points to an attribute list.

3110 `buffer`

3111 Points to a string buffer to receive the to receive the text representation of the
 3112 attribute list.

3113 `buflen`

3114 Specifies the length of the string buffer in bytes.

3115

3116 **Outputs**

3117

3118 `buffer`

3119 The buffer is filled with the text representation of the attribute list. The buffer
 3120 will always be set to something by this function (`buffer[0] = NULL` in cases of
 3121 an error).

3122

3123

Returns

3124

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

3125

3126

Example

3127

3128

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

3129

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

3130

```
char buffer[8192];
```

3131

```
...
```

3132

```
papiAttributeListToString(&attrs, buffer, sizeof(buffer));
```

3133

```
...
```

3134

```
papiAttributeListFree(attrs);
```

3135

3136

3137

See Also

3138

`papiAttributeListFromString`

3140 Chapter 7. Job API

3141 7.1. papiJobSubmit

3142 Description

3143 Submits a print job having the specified attributes to the specified printer. This
3144 interface copies the specified print files before returning to the caller (contrast to
3145 papiJobSubmitByReference).

3146 Syntax

3147

```
3148 papi_status_t papiJobSubmit(  
3149     papi_service_t      handle,  
3150     const char*         printer_name,  
3151     const papi_attribute_t** job_attributes,  
3152     const papi_job_ticket_t* job_ticket,  
3153     const char**        file_names,  
3154     papi_job_t*         job );  
3155
```

3156

3157 Inputs

3158

3159 handle

3160 Handle to the print service to use.

3161 printer_name

3162 Pointer to the name of the printer to which the job is to be submitted.

3163 job_attributes

3164 (optional) The list of attributes describing the job and how it is to be printed. If
3165 options are specified here and also in the job ticket data, the value specified
3166 here takes precedence. If this is NULL then only default attributes and
3167 (optionally) a job ticket is submitted with the job.

3168 job_ticket

3169 (optional) Pointer to structure specifying the job ticket. If this argument is
3170 NULL, then no job ticket is used with the job.

3171 Whether the implementation passes both the attributes and the job ticket to the
3172 server/printer, or merges them to some print protocol or internal
3173 implementation depends on the implementation.

3174 file_names

3175 NULL terminated list of pointers to names of files to print. If more than one
3176 file is specified, the files will be treated by the print system as separate
3177 "documents" for things like page breaks and separator sheets, but they will be
3178 scheduled and printed together as one job and the specified attributes will
3179 apply to all the files.

3180 These file names may contain absolute path names or relative path names
 3181 (relative to the current path). The implementation MUST copy the file contents
 3182 before returning.

3183

3184 **Outputs**

3185

3186 job

 The resulting job object representing the submitted job.

3188

3189 **Returns**

3190 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3191 value is returned.

3192 **Example**

3193

```

3194 #include "papi.h"
3195
3196 papi_status_t status;
3197 papi_service_t handle = NULL;
3198 const char* printer = "my-printer";
3199 const papi_attribute_t** attrs = NULL;
3200 const papi_job_ticket_t* ticket = NULL;
3201 const char* files[] = { "/etc/motd", NULL };
3202 papi_job_t job = NULL;
3203
3204
3205 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3206                           PAPI_ENCRYPT_IF_REQUESTED, NULL);
3207 if (status != PAPI_OK)
3208 {
3209     /* handle the error */
3210     ...
3211 }
3212
3213 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3214                            PAPI_STRING, 1, "test job");
3215 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3216                             PAPI_INTEGER, 1, 4);
3217
3218 status = papiJobSubmit(handle,
3219                       printer,
3220                       attrs,
3221                       ticket,
3222                       files,
3223                       &job);
3224 if (status != PAPI_OK)
3225 {
3226     fprintf(stderr, "papiJobSubmit failed: %s\n",
3227            papiStatusString(status));
3228     ...
3229 }
3230
3231 if (job != NULL)
3232 {
3233     /* look at the job object (maybe get the id) */
3234     papiJobFree(job);
3235 }
3236
3237 papiServiceDestroy(handle);
3238

```

3239

3240 **See Also**

3241 papiJobSubmitByReference, papiJobValidate, papiJobFree

3242 **7.2. papiJobSubmitByReference**3243 **Description**

3244 Submits a print job having the specified attributes to the specified printer. This
 3245 interface delays copying the specified print files as long as possible, ideally only
 3246 "pulling" the files when the printer is actually printing the job (contrast to
 3247 papiJobSubmit).

3248 **Syntax**

3249

```

3250 papi_status_t papiJobSubmitByReference(
3251     papi_service_t    handle,
3252     const char*       printer_name,
3253     const papi_attribute_t** job_attributes,
3254     const papi_job_ticket_t* job_ticket,
3255     const char**      file_names,
3256     papi_job_t*       job );
3257

```

3258

3259 **Inputs**

3260

3261 handle

3262 Handle to the print service to use.

3263 printer_name

3264 Pointer to the name of the printer to which the job is to be submitted.

3265 job_attributes

3266 (optional) The list of attributes describing the job and how it is to be printed. If
 3267 options are specified here and also in the job ticket data, the value specified
 3268 here takes precedence. If this is NULL then only default attributes and
 3269 (optionally) a job ticket is submitted with the job.

3270 job_ticket

3271 (optional) Pointer to structure specifying the job ticket. If this argument is
 3272 NULL, then no job ticket is used with the job.

3273 Whether the implementation passes both the attributes and the job ticket to the
 3274 server/printer, or merges them to some print protocol or internal
 3275 implementation depends on the implementation.

3276 file_names

3277 NULL terminated list of pointers to names of files to print. If more than one
 3278 file is specified, the files will be treated by the print system as separate
 3279 "documents" for things like page breaks and separator sheets, but they will be
 3280 scheduled and printed together as one job and the specified attributes will
 3281 apply to all the files.

3282 These file names may contain absolute path names, relative path names or
 3283 URIs ([RFC1738], [RFC2396]). The implementation SHOULD NOT copy the
 3284 referenced data unless (or until) it is no longer feasible to maintain the

3285 reference. Feasibility limitations may arise out of security issues, namespace
3286 issues, and/or protocol or printer limitations.

3287 Implementations MUST support the absolute path, relative path, and "file:"
3288 URI scheme. Use of other URI schemes could result in a
3289 PAPI_OPERATION_NOT_SUPPORTED error, depending on the
3290 implementation.

3291 The semantics explained in the preceding paragraphs allows for flexibility in
3292 the PAPI implementation. For example: (1) PAPI on top of a local service to
3293 maintain the reference for the life of the job, if the local service supports it. (2)
3294 PAPI on top of IPP to send a reference when the server can access the
3295 referenced data and copy it when it is not accessible to the server. (3) PAPI on
3296 top of network printing protocols that don't support references to copy the data
3297 on the way out to the remote server.

3298

3299 **Outputs**

3300

3301 job

The resulting job object representing the submitted job.

3303

3304 **Returns**

3305 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3306 value is returned.

3307 **Example**

3308

```

3309 #include "papi.h"
3310
3311 papi_status_t status;
3312 papi_service_t handle = NULL;
3313 const char* printer = "my-printer";
3314 const papi_attribute_t** attrs = NULL;
3315 const papi_job_ticket_t* ticket = NULL;
3316 const char* files[] = { "http://foo.bar.org/docs/glop.pdf", NULL };
3317 papi_job_t job = NULL;
3318
3319
3320 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3321     PAPI_ENCRYPT_IF_REQUESTED, NULL);
3322 if (status != PAPI_OK)
3323 {
3324     /* handle the error */
3325     ...
3326 }
3327
3328 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3329     PAPI_STRING, 1, "test job");
3330 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3331     PAPI_INTEGER, 1, 4);
3332
3333 status = papiJobSubmitByReference(handle,
3334     printer,
3335     attrs,
3336     ticket,
3337     files,
3338     &job);
3339
3340 if (status != PAPI_OK)
3341 {
3342     fprintf(stderr, "papiJobSubmitByReference failed: %s\n",
3343         papiStatusString(status));
3344     ...
3345 }
3346
3347 if (job != NULL)
3348 {

```

```

3347         /* look at the job object (maybe get the id) */
3348     }
3349     papiJobFree(job);
3350 }
3351
3352 papiServiceDestroy(handle);
3353

```

3354

3355 **See Also**

3356 papiJobSubmit, papiJobValidate, papiJobFree

3357 **7.3. papiJobValidate**3358 **Description**

3359 Validates the specified job attributes against the specified printer. This function can
 3360 be used to validate the capability of a print object to accept a specific combination of
 3361 attributes.

3362 **Syntax**

3363

```

3364 papi_status_t papiJobValidate(
3365     papi_service_t      handle,
3366     const char*         printer_name,
3367     const papi_attribute_t** job_attributes,
3368     const papi_job_ticket_t* job_ticket,
3369     const char**        file_names,
3370     papi_job_t*         job );
3371

```

3372

3373 **Inputs**

3374

3375 handle

3376 Handle to the print service to use.

3377 printer_name

3378 Pointer to the name of the printer against which the job is to be validated.

3379 job_attributes

3380 (optional) The list of attributes describing the job and how it is to be printed. If
 3381 options are specified here and also in the job ticket data, the value specified
 3382 here takes precedence. If this is NULL then only default attributes and
 3383 (optionally) a job ticket is submitted with the job.

3384 job_ticket

3385 (optional) Pointer to structure specifying the JDF job ticket. If this argument is
 3386 NULL, then no job ticket is used with the job.

3387 file_names

3388 NULL terminated list of pointers to names of files to validate.

3389

3390

Outputs

3391

3392 job

3393

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

3394

3395

Returns

3396

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

3397

3398

Example

3399

3400

```

3401 #include "papi.h"
3402
3403 papi_status_t status;
3404 papi_service_t handle = NULL;
3405 const char* printer = "my-printer";
3406 const papi_attribute_t** attrs = NULL;
3407 const papi_job_ticket_t* ticket = NULL;
3408 const char* files[] = { "/etc/motd", NULL };
3409 papi_job_t job = NULL;
3410
3411 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3412                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
3413 if (status != PAPI_OK)
3414 {
3415     /* handle the error */
3416     ...
3417 }
3418
3419 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3420                           PAPI_STRING, 1, "test job");
3421 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3422                             PAPI_INTEGER, 1, 4);
3423
3424 status = papiJobValidate(handle,
3425                          printer,
3426                          attrs,
3427                          ticket,
3428                          files,
3429                          &job);
3430 if (status != PAPI_OK)
3431 {
3432     fprintf(stderr, "papiJobValidate failed: %s\n",
3433            papiStatusString(status));
3434     ...
3435 }
3436
3437 if (job != NULL)
3438 {
3439     ...
3440     papiJobFree(job);
3441 }
3442
3443 papiServiceDestroy(handle);

```

3444

3445

See Also

3446

papiJobSubmit, papiJobFree

3447

7.4. papiJobStreamOpen

3448

Description

3449

Opens a print job and an associated stream of print data to be sent to the specified printer. After calling this function papiJobStreamWrite can be called (repeatedly) to

3450

3451 write the print data to the stream, and then `papiJobStreamClose` is called to
 3452 complete the submission of the print job.

3453 After this function is called successfully, `papiJobStreamClose` must eventually be
 3454 called to close the stream (this includes all error paths).

3455 **Syntax**

3456

```

3457 papi_status_t papiJobStreamOpen(
3458             papi_service_t      handle,
3459             const char*         printer_name,
3460             const papi_attribute_t** job_attributes,
3461             const papi_job_ticket_t* job_ticket,
3462             papi_stream_t*      stream );
3463
```

3464

3465 **Inputs**

3466

3467 `handle`

3468 Handle to the print service to use.

3469 `printer_name`

3470 Pointer to the name of the printer to which the job is to be submitted.

3471 `job_attributes`

3472 (optional) The list of attributes describing the job and how it is to be printed.
 3473 See `job_attributes` argument for `papiJobSubmit` for description.

3474 `job_ticket`

3475 (optional) Pointer to structure specifying the job ticket. See `job_ticket` argument
 3476 for `papiJobSubmit` for description.

3477

3478 **Outputs**

3479

3480 `stream`

3481 The resulting stream object to which print data can be written.

3482

3483 **Returns**

3484 If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate failure
 3485 value is returned.

3486 **Example**

3487

```

3488 #include "papi.h"
3489
3490 papi_status_t status;
3491 papi_service_t handle = NULL;
3492 const char* printer = "my-printer";
```

```

3493     const papi_attribute_t** attrs = NULL;
3494     const papi_job_ticket_t* ticket = NULL;
3495     papi_stream_t stream = NULL;
3496     papi_job_t job = NULL;
3497     char buffer[4096];
3498     int buflen = 0;
3499
3500     status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3501                               PAPI_ENCRYPT_IF_REQUESTED, NULL);
3502     if (status != PAPI_OK)
3503     {
3504         /* handle the error */
3505         ...
3506     }
3507
3508     papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3509                               PAPI_STRING, 1, "test job");
3510     papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3511                                PAPI_INTEGER, 1, 4);
3512
3513     /* Open the print job stream */
3514     status = papiJobStreamOpen(handle,
3515                                printer,
3516                                attrs,
3517                                ticket,
3518                                &stream);
3519     if (status != PAPI_OK)
3520     {
3521         fprintf(stderr, "papiJobStreamOpen failed: %s\n",
3522                papiStatusString(status));
3523         ...
3524     }
3525
3526     /* Write all the print job data */
3527     while(print_data_remaining)
3528     {
3529         /* Generate the print data */
3530         ...
3531         /* Write the print data */
3532         status = papiJobStreamWrite(handle
3533                                     stream,
3534                                     buffer,
3535                                     buflen);
3536         if (status != PAPI_OK)
3537         {
3538             fprintf(stderr, "papiJobStreamWrite failed: %s\n",
3539                    papiStatusString(status));
3540             ...
3541         }
3542     }
3543
3544     /* Close the print job stream */
3545     status = papiJobStreamClose(handle, stream, &job);
3546     if (status != PAPI_OK)
3547     {
3548         fprintf(stderr, "papiJobStreamClose failed: %s\n",
3549                papiStatusString(status));
3550         ...
3551     }
3552
3553     papiJobFree(job);
3554     papiServiceDestroy(handle);
3555

```

3556

3557

See Also

3558

papiJobStreamWrite, papiJobStreamClose

3559

7.5. papiJobStreamWrite

3560

Description

3561

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

3562

3563

Syntax

3564

3565

papi_status_t papiJobStreamWrite(

```

3566         papi_service_t     handle,
3567         papi_stream_t      stream,
3568         const char*        buffer,
3569         const int          buflen );
3570

```

3571

3572 **Inputs**

3573

3574 handle

3575 Handle to the print service to use.

3576 stream

3577 The open stream object to which print data is written.

3578 buffer

3579 Pointer to the buffer of print data to write.

3580 buflen

3581 The number of bytes to write.

3582

3583 **Outputs**

3584 none

3585 **Returns**

3586 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3587 value is returned.

3588 **Example**

3589 See papiJobStreamOpen

3590 **See Also**

3591 papiJobStreamOpen, papiJobStreamClose

3592 7.6. papiJobStreamClose

3593 **Description**

3594 Closes the specified open job stream and completes submission of the job (if there
3595 were no previous errors returned from papiJobSubmitWrite). The open job stream
3596 must have been obtained by a successful call to papiJobStreamOpen.

3597 **Syntax**

3598

```

3599 papi_status_t papiJobStreamClose(
3600         papi_service_t     handle,
3601         papi_stream_t      stream,
3602         papi_job_t*        job );
3603

```

3604

3605 **Inputs**

3606

3607 handle

3608 Handle to the print service to use.

3609 stream

3610 The open stream object to which print data was written.

3611

3612 **Outputs**

3613

3614 job

3615 The resulting job object representing the submitted job.

3616

3617 **Returns**3618 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3619 value is returned.3620 **Example**

3621 See papiJobStreamOpen

3622 **See Also**

3623 papiJobStreamOpen, papiJobStreamWrite

3624 **7.7. papiJobQuery**3625 **Description**

3626 Queries some or all the attributes of the specified job object.

3627 **Syntax**

3628

```

3629 papi_status_t papiJobQuery(
3630             papi_service_t   handle,
3631             const char*       printer_name,
3632             const int32_t     job_id,
3633             const char*       requested_attrs[],
3634             papi_job_t*       job );
3635

```

3636

3637 **Inputs**

3638

3639 handle

3640 Handle to the print service to use.

3641 printer_name

3642 Pointer to the name or URI of the printer to which the job was submitted.

3643 job_id

3644 The ID number of the job to be queried.

3645 requested_attrs

3646 NULL terminated array of attributes to be queried. If NULL is passed then all
 3647 available attributes are queried. (NOTE: The job may return more attributes
 3648 than you requested. This is merely an advisory request that may reduce the
 3649 amount of data returned if the printer/server supports it.)

3650

3651 **Outputs**

3652

3653 job

3654 The returned job object containing the requested attributes.

3655

3656 **Returns**

3657 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3658 value is returned.

3659 **Example**

3660

```

3661 #include "papi.h"
3662
3663 papi_status_t status;
3664 papi_service_t handle = NULL;
3665 const char* printer_name = "my-printer";
3666 papi_job_t job = NULL;
3667 int32_t job_id = 12;
3668 const char* job_attrs[] =
3669 {
3670     "job-id",
3671     "job-name",
3672     "job-originating-user-name",
3673     "job-state",
3674     "job-state-reasons",
3675     NULL
3676 };
3677 ...
3678 status = papiServiceCreate(&handle,
3679                             NULL,
3680                             NULL,
3681                             NULL,
3682                             NULL,
3683                             PAPI_ENCRYPT_NEVER,
3684                             NULL);
3685
3686 if (status != PAPI_OK)
3687 {
3688     /* handle the error */
3689     ...
3690 }
3691
3692 status = papiJobQuery(handle,
3693                       printer_name,
3694                       job_id,
3695                       job_attrs,
3696                       &job);
3697
3698 if (status != PAPI_OK)
3699 {
3700     /* handle the error */
3701     fprintf(stderr, "papiJobQuery failed: %s\n",
3702             papiServiceGetStatusMessage(handle));
3703     ...
3704 }
3705
3706 if (job != NULL)
3707 {
3708     /* process the job */
3709     ...

```



```

3708     papiJobFree(job);
3709 }
3710
3711 papiServiceDestroy(handle);
3712

```

3713

3714 **See Also**

3715 papiJobFree, papiPrinterListJobs, papiJobModify

3716 **7.8. papiJobModify**3717 **Description**

3718 Modifies some or all the attributes of the specified job object.

3719 **Syntax**

3720

```

3721 papi_status_t papiJobModify(
3722     papi_service_t    handle,
3723     const char*       printer_name,
3724     const int32_t     job_id,
3725     const papi_attribute_t** attrs,
3726     papi_job_t*      job );
3727

```

3728

3729 **Inputs**

3730

3731 handle

3732 Handle to the print service to use.

3733 printer_name

3734 Pointer to the name or URI of the printer to which the job was submitted.

3735 job_id

3736 The ID number of the job to be modified.

3737 attrs

3738 Attributes to be modified. Any attributes not specified are left unchanged.

3739

3740 **Outputs**

3741

3742 job

3743 The modified job object.

3744

3745 **Returns**

3746 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3747 value is returned.

3748

Example

3749

```

3750 #include "papi.h"
3751
3752 papi_status_t status;
3753 papi_service_t handle = NULL;
3754 const char* printer_name = "my-printer";
3755 papi_job_t job = NULL;
3756 int32_t job_id = 12;
3757 papi_attribute_t** attrs = NULL;
3758 ...
3759 status = papiServiceCreate(&handle,
3760                             NULL,
3761                             NULL,
3762                             NULL,
3763                             NULL,
3764                             PAPI_ENCRYPT_NEVER,
3765                             NULL);
3766
3767 if (status != PAPI_OK)
3768 {
3769     /* handle the error */
3770     ...
3771 }
3772
3773 papiAttributeListAddInteger(&attrs,
3774                             PAPI_EXCL,
3775                             "copies",
3776                             3);
3777
3778 status = papiJobModify(handle,
3779                        printer_name,
3780                        job_id,
3781                        attrs,
3782                        &job);
3783
3784 if (status != PAPI_OK)
3785 {
3786     /* handle the error */
3787     fprintf(stderr, "papiJobModify failed: %s\n",
3788            papiServiceGetStatusMessage(handle));
3789     ...
3790 }
3791
3792 if (job != NULL)
3793 {
3794     /* process the job */
3795     ...
3796     papiJobFree(job);
3797 }
3798
3799 papiServiceDestroy(handle);

```

3799

3800

See Also

3801

papiJobQuery, papiJobFree, papiPrinterListJobs

7.9. papiJobCancel

3803

Description

3804

Cancel the specified print job.

3805

Syntax

3806

```

3807 papi_status_t papiJobCancel(
3808             papi_service_t handle,
3809             const char* printer_name,
3810             const int32_t job_id );
3811

```

3812

3813 **Inputs**

3814

3815 handle

3816 Handle to the print service to use.

3817 printer_name

3818 Pointer to the name or URI of the printer to which the job was submitted.

3819 job_id

3820 The ID number of the job to be cancelled.

3821

3822 **Outputs**

3823 none

3824 **Returns**3825 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3826 value is returned.3827 **Example**

3828

```

3829 #include "papi.h"
3830
3831 papi_status_t status;
3832 papi_service_t handle = NULL;
3833 const char* printer_name = "my-printer";
3834 int32_t job_id = 12;
3835 ...
3836 status = papiServiceCreate(&handle,
3837                            NULL,
3838                            NULL,
3839                            NULL,
3840                            NULL,
3841                            PAPI_ENCRYPT_NEVER,
3842                            NULL);
3843
3844 if (status != PAPI_OK)
3845 {
3846     /* handle the error */
3847     ...
3848 }
3849
3850 status = papiJobCancel(handle,
3851                        printer_name,
3852                        job_id);
3853
3854 if (status != PAPI_OK)
3855 {
3856     /* handle the error */
3857     fprintf(stderr, "papiJobCancel failed: %s\n",
3858            papiServiceGetStatusMessage(handle));
3859     ...
3860 }
3861 papiServiceDestroy(handle);

```

3862

3863 **See Also**

3864 papiPrinterListJobs, papiPrinterPurgeJobs

3865 **7.10. papiJobHold**3866 **Description**

3867 Holds the specified print job and prevents it from being scheduled for printing.
 3868 This operation is optional and may not be supported by all printers/servers. Use
 3869 papiJobRelease to undo the effects of this operation, or specify the hold_until
 3870 argument to automatically release the job at a specific time.

3871 **Syntax**

3872

```

3873 papi_status_t papiJobHold(
3874             papi_service_t   handle,
3875             const char*      printer_name,
3876             const int32_t    job_id,
3877             const char*      hold_until,
3878             const time_t*    hold_until_time );
3879

```

3880

3881 **Inputs**

3882

3883 handle

3884 Handle to the print service to use.

3885 printer_name

3886 Pointer to the name or URI of the printer to which the job was submitted.

3887 job_id

3888 The ID number of the job to be held.

3889 hold_until

3890 (optional) Specifies the time when the job will be automatically released for
 3891 printing. If NULL, the job is held until explicitly released by calling
 3892 papiJobRelease. If specified, the value must be one of the strings "indefinite"
 3893 (same effect as passing NULL), "day-time", "evening", "night", "weekend",
 3894 "second-shift", "third-shift", or "timed". For values other than "indefinite" and
 3895 "timed", the printer/server must define exact times associated with these
 3896 values and it may make these associations configurable. If "timed" is specified,
 3897 then the hold_until_time argument is used.

3898 hold_until_time

3899 (optional) Specifies the time when the job will be automatically released for
 3900 printing. This argument is ignored unless "timed" is passed as the hold_until
 3901 argument.

3902

3903 **Outputs**

3904 none

3905

Returns

3906

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

3907

3908

Example

3909

```

3910 #include "papi.h"
3911
3912 papi_status_t status;
3913 papi_service_t handle = NULL;
3914 const char* printer_name = "my-printer";
3915 int32_t job_id = 12;
3916 ...
3917 status = papiServiceCreate(&handle,
3918                          NULL,
3919                          NULL,
3920                          NULL,
3921                          NULL,
3922                          PAPI_ENCRYPT_NEVER,
3923                          NULL);
3924
3925 if (status != PAPI_OK)
3926 {
3927     /* handle the error */
3928     ...
3929 }
3930
3931 status = papiJobHold(handle,
3932                    printer_name,
3933                    job_id,
3934                    NULL,
3935                    NULL);
3936
3937 if (status != PAPI_OK)
3938 {
3939     /* handle the error */
3940     fprintf(stderr, "papiJobHold failed: %s\n",
3941            papiServiceGetStatusMessage(handle));
3942     ...
3943 }
3944 papiServiceDestroy(handle);

```

3945

See Also

3946

papiJobRelease

3947

7.11. papiJobRelease

3948

3949

Description

3950

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

3951

3952

3953

Syntax

3954

```

3955 papi_status_t papiJobRelease(
3956             papi_service_t      handle,
3957             const char*         printer_name,
3958             const int32_t       job_id );
3959

```

3960

3961

Inputs

3962

3963 handle
 3964 Handle to the print service to use.
 3965 printer_name
 3966 Pointer to the name or URI of the printer to which the job was submitted.
 3967 job_id
 3968 The ID number of the job to be released.
 3969

3970 **Outputs**

3971 none

3972 **Returns**

3973 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3974 value is returned.

3975 **Example**

3976

```

3977 #include "papi.h"
3978
3979 papi_status_t status;
3980 papi_service_t handle = NULL;
3981 const char* printer_name = "my-printer";
3982 int32_t job_id = 12;
3983 ...
3984 status = papiServiceCreate(&handle,
3985                          NULL,
3986                          NULL,
3987                          NULL,
3988                          NULL,
3989                          PAPI_ENCRYPT_NEVER,
3990                          NULL);
3991
3992 if (status != PAPI_OK)
3993 {
3994     /* handle the error */
3995     ...
3996 }
3997
3998 status = papiJobRelease(handle,
3999                       printer_name,
4000                       job_id);
4001
4002 if (status != PAPI_OK)
4003 {
4004     /* handle the error */
4005     fprintf(stderr, "papiJobRelease failed: %s\n",
4006           papiServiceGetStatusMessage(handle));
4007     ...
4008 }
4009 papiServiceDestroy(handle);

```

4010

4011 **See Also**

4012 papiJobHold

4013 **7.12. papiJobRestart**

4014 **Description**

4015 Restarts a job that was retained after processing. If and how a job is retained after
 4016 processing is implementation-specific and is not covered by this API. This operation
 4017 is optional and may not be supported by all printers/servers.

4018 **Syntax**

4019

```

4020     papi_status_t papiJobRestart(
4021                 papi_service_t   handle,
4022                 const char*       printer_name,
4023                 const int32_t     job_id );
4024

```

4025

4026 **Inputs**

4027

4028 `handle`4029 `Handle to the print service to use.`4030 `printer_name`4031 `Pointer to the name or URI of the printer to which the job was submitted.`4032 `job_id`4033 `The ID number of the job to be restarted.`

4034

4035 **Outputs**4036 `none`4037 **Returns**

4038 `If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure`
 4039 `value is returned.`

4040 **Example**

4041

```

4042     #include "papi.h"
4043
4044     papi_status_t status;
4045     papi_service_t handle = NULL;
4046     const char* printer_name = "my-printer";
4047     int32_t job_id = 12;
4048     ...
4049     status = papiServiceCreate(&handle,
4050                               NULL,
4051                               NULL,
4052                               NULL,
4053                               NULL,
4054                               PAPI_ENCRYPT_NEVER,
4055                               NULL);
4056
4057     if (status != PAPI_OK)
4058     {
4059         /* handle the error */
4060         ...
4061     }
4062
4063     status = papiJobRestart(handle,
4064                             printer_name,
4065                             job_id);
4066
4067     if (status != PAPI_OK)
4068     {
4069         /* handle the error */
4070         fprintf(stderr, "papiJobRestart failed: %s\n",
4071                papiServiceGetStatusMessage(handle));
4072         ...
4073     }
4074
4075     papiServiceDestroy(handle);

```

4074

4075

4076

See Also

4077

papiPrinterListJobs

4078

7.13. papiJobGetAttributeList

4079

Description

4080

Get the attribute list associated with a job object.

4081

Syntax

4082

4083

4084

4085

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

4086

4087

Inputs

4088

4089

job

4090

Handle of the job object.

4091

4092

Outputs

4093

none

4094

Returns

4095

Pointer to the attribute list associated with the job object.

4096

Example

4097

4098

4099

4100

4101

4102

4103

4104

4105

4106

4107

4108

4109

4110

4111

4112

4113

4114

4115

4116

4117

4118

4119

4120

4121

4122

4123

4124

4125

4126

4127

```
#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
papi_job_t job = NULL;
papi_attribute_list* attrs = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);

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

status = papiJobQuery(handle,
                     printer_name,
                     67,
                     NULL,
                     &job);

if (status != PAPI_OK)
{
    /* handle the error */
    fprintf(stderr, "papiJobQuery failed: %s\n",
```



```

4128         papiServiceGetStatusMessage(handle));
4129     ...
4130 }
4131
4132 if (job != NULL)
4133 {
4134     /* process the job object */
4135     attrs = papiJobGetAttributeList(job);
4136     ...
4137     papiJobFree(job);
4138 }
4139
4140 papiServiceDestroy(handle);
4141

```

4142

4143 **See Also**

4144 papiPrinterListJobs, papiJobQuery

4145 **7.14. papiJobGetPrinterName**4146 **Description**

4147 Get the printer name associated with a job object.

4148 **Syntax**

4149

```

4150 char* papiJobGetPrinterName(
4151         papi_job_t job );
4152

```

4153

4154 **Inputs**

4155

4156 job

4157 Handle of the job object.

4158

4159 **Outputs**

4160 none

4161 **Returns**

4162 Pointer to the printer name associated with the job object.

4163 **Example**

4164

```

4165 #include "papi.h"
4166
4167 char* printer_name = NULL;
4168 papi_job_t job = NULL;
4169 ...
4170 if (job != NULL)
4171 {
4172     /* process the job object */
4173     printer_name = papiJobGetPrinterName(job);
4174     ...
4175     papiJobFree(job);
4176 }
4177

```

4178

4179 **See Also**
4180 papiPrinterListJobs, papiJobQuery

4181 **7.15. papiJobGetId**

4182 **Description**

4183 Get the job ID associated with a job object.

4184 **Syntax**

4185

```
4186           int32_t papiJobGetId(  
4187                               papi_job_t     job );  
4188
```

4189

4190 **Inputs**

4191

4192 job

4193 Handle of the job object.

4194

4195 **Outputs**

4196 none

4197 **Returns**

4198 The job ID associated with the job object.

4199 **Example**

4200

```
4201           #include "papi.h"  
4202  
4203           int32_t job_id;  
4204           papi_job_t job = NULL;  
4205           ...  
4206           if (job != NULL)  
4207           {  
4208               /* process the job object */  
4209               job_id = papiJobGetId(job);  
4210               ...  
4211               papiJobFree(job);  
4212           }  
4213
```

4214

4215 **See Also**

4216 papiPrinterListJobs, papiJobQuery

4217 **7.16. papiJobGetJobTicket**

4218 **Description**

4219 Get the job ticket associated with a job object.

4220 **Syntax**

4221

```

4222     papi_job_ticket_t* papiJobGetJobTicket(
4223         papi_job_t     job );
4224

```

4225

4226 **Inputs**

4227

4228 job

4229 Handle of the job object.

4230

4231 **Outputs**

4232 none

4233 **Returns**

4234 Pointer to the job ticket associated with the job object.

4235 **Example**

4236

```

4237 #include "papi.h"
4238
4239 papi_job_ticket_t* job_ticket = NULL;
4240 papi_job_t job = NULL;
4241 ...
4242 if (job != NULL)
4243 {
4244     /* process the job object */
4245     job_ticket = papiJobGetJobTicket(job);
4246     ...
4247     papiJobFree(job);
4248 }
4249

```

4250

4251 **See Also**

4252 papiPrinterListJobs, papiJobQuery

4253 **7.17. papiJobFree**4254 **Description**

4255 Free a job object.

4256 **Syntax**

4257

```

4258 void papiJobFree(
4259     papi_job_t     job );
4260

```

4261

4262 **Inputs**

4263

4264 job

4265 Handle of the job object to free.

4266

4267

Outputs

4268

none

4269

Returns

4270

none

4271

Example

4272

4273

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

4274

```
papi_status_t status;
```

4275

```
papi_service_t handle = NULL;
```

4276

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

4277

```
papi_job_t job = NULL;
```

4278

```
...
```

4279

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

4280

```
NULL,
```

4281

```
NULL,
```

4282

```
NULL,
```

4283

```
NULL,
```

4284

```
NULL,
```

4285

```
PAPI_ENCRYPT_NEVER,
```

4286

```
NULL);
```

4287

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

4288

```
{
```

4289

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

4290

```
    ...
```

4291

```
}
```

4292

```
status = papiJobQuery(handle,
```

4293

```
    printer_name,
```

4294

```
    12,
```

4295

```
    &job);
```

4296

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

4297

```
{
```

4298

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

4299

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

4300

```
        papiServiceGetStatusMessage(handle));
```

4301

```
    ...
```

4302

```
}
```

4303

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

4304

```
{
```

4305

```
    /* process the job object */
```

4306

```
    ...
```

4307

```
    papiJobFree(job);
```

4308

```
}
```

4309

```
papiServiceDestroy(handle);
```

4310

4311

4312

4314

See Also

4315

papiJobQuery

4316

7.18. papiJobListFree

4317

Description

4318

Free a list of job objects.

4319

Syntax

4320

```
void papiJobListFree(
```

4321

```
    papi_job_t*    jobs );
```

4322

4323

4324

4325

4326 **Inputs**

4327

4328 jobs

4329 Pointer to the job object list to free.

4330

4331 **Outputs**

4332 none

4333 **Returns**

4334 none

4335 **Example**

4336

```

4337 #include "papi.h"
4338
4339 papi_status_t status;
4340 papi_service_t handle = NULL;
4341 const char* printer_name = "my-printer";
4342 papi_job_t* jobs = NULL;
4343 ...
4344 status = papiServiceCreate(&handle,
4345                          NULL,
4346                          NULL,
4347                          NULL,
4348                          NULL,
4349                          PAPI_ENCRYPT_NEVER,
4350                          NULL);
4351
4352 if (status != PAPI_OK)
4353 {
4354     /* handle the error */
4355     ...
4356 }
4357
4358 status = papiPrinterListJobs(handle,
4359                             printer_name,
4360                             NULL,
4361                             0, 0, 0,
4362                             &jobs);
4363
4364 if (status != PAPI_OK)
4365 {
4366     /* handle the error */
4367     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
4368            papiServiceGetStatusMessage(handle));
4369     ...
4370 }
4371
4372 if (jobs != NULL)
4373 {
4374     /* process the job objects */
4375     ...
4376     papiJobListFree(jobs);
4377 }
4378
4379 papiServiceDestroy(handle);

```

4379

4380 **See Also**

4381 papiPrinterListJobs

4382 Chapter 8. Miscellaneous API

4383 8.1. papiStatusString

4384 Description

4385 Get a status string for the specified papi_status_t. The status message returned
4386 from this function may be less detailed than the status message returned from
4387 papiServiceGetStatusMessage (if the print service supports returning more detailed
4388 error messages).

4389 The returned message will be localized in the language of the submitter of the
4390 requestor.

4391 Syntax

4392

```
4393 char* papiStatusString(  
4394     const papi_status_t status );  
4395
```

4396

4397 Inputs

4398

4399 status

4400 The status value to convert to a status string.

4401

4402 Outputs

4403 none

4404 Returns

4405 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
4406 value is returned.

4407 Example

4408

```
4409 #include "papi.h"  
4410  
4411 papi_status_t status;  
4412 ...  
4413 fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));  
4414
```

4415

4416 See Also

4417 papiServiceGetStatusMessage

4418 Chapter 9. Attributes

4419 For a summary of the IPP attributes which can be used with the PAPI interface, see:
4420 <ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf>

4421 9.1. Extension Attributes

4422 The following attributes are not currently defined by IPP, but may be used with
4423 this API.

4424 9.1.1. job-ticket-formats-supported

4425 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats
4426 that are supported by the printer. If this attribute is not present, it is assumed that
4427 the printer does not support any job ticket formats.

4428

4429 9.2. Required Job Attributes

4430 The following job attributes *must* be supported to comply with this API standard.
4431 These attributes may be supported by the underlying print server directly, or they
4432 may be mapped by the PAPI library.

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

4433

4434 9.3. Required Printer Attributes

4435 The following printer attributes *must* be supported to comply with this API
4436 standard. These attributes may be supported by the underlying print server
4437 directly, or they may be mapped by the PAPI library.

charset-configured
charset-supported
compression-supported
document-format-default
document-format-supported
generated-natural-language-supported
natural-language-configured
operations-supported
pdl-override-supported
printer-is-accepting-jobs
printer-name
printer-state

4438 printer-state-reasons
 printer-up-time
 printer-uri-supported
 queued-job-count
 uri-authentication-supported
 uri-security-supported

4439 **9.4. IPP Attribute Type Mapping**

4440 The following table maps IPP to PAPI attribute value types:

IPP Type	PAPI Type
boolean	PAPI_BOOLEAN
charset	PAPI_STRING
collection	PAPI_COLLECTION
dateTime	PAPI_DATETIME
enum	PAPI_INTEGER (with C enum values)
integer	PAPI_INTEGER
keyword	PAPI_STRING
mimeMediaType	PAPI_STRING
name	PAPI_STRING
naturalLanguage	PAPI_STRING
octetString	not yet supported
rangeOfInteger	PAPI_RANGE
resolution	PAPI_RESOLUTION
text	PAPI_STRING
uri	PAPI_STRING
uriScheme	PAPI_STRING
1setOf X	C array

4441

4442 Appendix A. Attribute List Text Representation

4443 A.1. ABNF Definition

4444 The following ABNF definition [RFC2234] describes the syntax of PAPI attributes
4445 encoded as text options:

```
4446 OPTION-STRING = [OPTION] *(1*WC OPTION) *WC
4447
4448 OPTION        = TRUEOPTION / FALSEOPTION / VALUEOPTION
4449
4450 TRUEOPTION    = NAME
4451
4452 FALSEOPTION   = "no" NAME
4453
4454 VALUEOPTION   = NAME "=" VALUE *( "," VALUE )
4455
4456 NAME         = 1*NAMECHAR
4457
4458 NAMECHAR     = DIGIT / ALPHA / "-" / "_" / "."
4459
4460 VALUE        = BOOLVALUE / COLVALUE / DATEVALUE / NUMBEVALUE / QUOTEDVALUE /
4461              RANGEVALUE / RESVALUE / STRINGVALUE
4462
4463 BOOLVALUE    = "yes" / "no" / "true" / "false"
4464
4465 COLVALUE     = "{" OPTION-STRING "}"
4466
4467 DATEVALUE    = HOUR MINUTE [ SECOND ] / YEAR MONTH DAY /
4468              YEAR MONTH DAY HOUR MINUTE [ SECOND ]
4469
4470 YEAR        = 4DIGIT
4471
4472 MONTH       = "0" %x31-39 / "10" / "11" / "12"
4473
4474 DAY         = %x30-32 DIGIT / "1" DIGIT / "2" DIGIT / "30" / "31"
4475
4476 HOUR        = %x30-31 DIGIT / "1" DIGIT / "20" / "21" / "22" / "23"
4477
4478 MINUTE      = %x30-35 DIGIT
4479
4480 SECOND      = %x30-35 DIGIT
4481
4482 NUMBEVALUE  = 1*DIGIT / "-" 1*DIGIT / "+" 1*DIGIT
4483
4484 QUOTEDVALUE = DQUOTE *QUOTEDCHAR DQUOTE / SQUOTE *QUOTEDCHAR SQUOTE
4485
4486 QUOTEDCHAR  = %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
4487              %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
4488              %x5D-7E / %xA0-FF
4489
4490 OCTALDIGIT  = %x30-37
4491
4492 RANGEVALUE  = 1*DIGIT "-" 1*DIGIT
4493
4494 RESVALUE    = 1*DIGIT [ "x" 1*DIGIT ] ("dpi" / "dpc")
4495
4496 STRINGVALUE = 1*STRINGCHAR
4497
4498 STRINGCHAR  = %x5C %x20 / %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
4499              %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
4500              %x5D-7E / %xA0-FF
4501
4502 SQUOTE      = %x27
4503
```

4504 A.2. Examples

4505 The following example strings illustrate the format of text options:

4506 Boolean Attributes:

```
4507 foo
4508 nofoo
4509 foo=false
4510 foo=true
4511 foo=no
4512 foo=yes
4513
```

Appendix A. Attribute List Text Representation

4514 **Collection Attributes:**
4515 media-col={media-size={x-dimension=123 y-dimension=456}}
4516

4517 **Integer Attributes:**
4518 copies=123
4519 hue=-123
4520

4521 **String Attributes:**
4522 job-sheets=standard
4523 job-sheets=standard, standard
4524 media=na-custom-foo.8000-10000
4525 job-name=John\'s\ Really\040Nice\ Document
4526

4527 **String Attributes (quoted):**
4528 job-name="John\'s Really Nice Document"
4529 document-name='Another \'Word\042 document.doc'
4530

4531 **Range Attributes:**
4532 page-ranges=1-5
4533 page-ranges=1-2,5-6,101-120
4534

4535 **Date Attributes:**
4536 job-hold-until-datetime=1234
4537 job-hold-until-datetime=123456
4538 job-hold-until-datetime=20020904
4539 job-hold-until-datetime=200209041234
4540 job-hold-until-datetime=20020904123456
4541

4542 **Resolution Attributes:**
4543 resolution=360dpi
4544 resolution=720x360dpi
4545 resolution=1000dpc
4546

4547 **Multiple Attributes:**
4548 job-sheets=standard page-ranges=1-2,5-6,101-120 resolution=360dpi
4549

4550 **Appendix B. References**

4551 **B.1. Internet Printing Protocol (IPP)**

- 4552 [RFC2911] T. Hastings, R. Herriot, R. deBry, S. Isaacson, and P. Powell August 1998
4553 *Internet Printing Protocol/1.1: Model and Semantics* (Obsoletes 2566)
- 4554 [RFC3380] T. Hastings, R. Herriot, C. Kugler, and H. Lewis September 2002 *Internet*
4555 *Printing Protocol (IPP): Job and Printer Set Operations*
- 4556 [RFC3381] T. Hastings, H. Lewis, and R. Bergman September 2002 *Internet Printing*
4557 *Protocol (IPP): Job Progress Attributes*
- 4558 [RFC3382] R. deBry, T. Hastings, R. Herriot, K. Ocke, and P. Zehler September 2002
4559 *Internet Printing Protocol (IPP): The 'collection' attribute syntax*
- 4560 [5100.2] T. Hastings and R. Bergman IEEE-ISTO 5100.2 February 2001 *Internet*
4561 *Printing Protocol (IPP): output-bin attribute extension*
- 4562 [5100.3] T. Hastings and K. Ocke IEEE-ISTO 5100.3 February 2001 *Internet Printing*
4563 *Protocol (IPP): Production Printing Attributes*
- 4564 [5100.4] R. Herriot and K. Ocke IEEE-ISTO 5100.4 February 2001 *Internet Printing*
4565 *Protocol (IPP): Override Attributes for Documents and Pages*
- 4566 [ops-set2] C. Kugler, T. Hastings, and H. Lewis July 2001 *Internet Printing Protocol*
4567 *(IPP): Job and Printer Administrative Operations*

4568 **B.2. Job Ticket**

- 4569 [jdf] CIP4 Organization April 2002 *Job Definition Format (JDF) Specification Version*
4570 *1.1*

4571 **B.3. Printer Working Group (PWG)**

- 4572 [PWGSemMod] P. Zehler and Albright September 2002 *Printer Working Group*
4573 *(PWG): Semantic Model*

4574 **B.4. Other**

- 4575 [RFC1738] T. Berners-Lee, L. Masinter, and M. McCahill December 1994 *Uniform*
4576 *Resource Locators (URL)* (Updated by RFC1808, RFC2368, RFC2396)
- 4577 [RFC2234] D. Crocker and P. Overell November 1997 *Augmented BNF for Syntax*
4578 *Specifications: ABNF*
- 4579 [RFC2396] T. Berners-Lee, R. Fielding, and L. Masinter August 1998 *Uniform*
4580 *Resource Locators (URL): Generic Syntax* (Updates RFC1808, RFC1738)

4581 **Appendix C. Change History**

4582 **Version 0.6 (September 20, 2002)**

4583

- 4584 • Made explanation of requested_attrs in papiPrintersList the same as it is for
4585 papiPrinterQuery.
- 4586 • Moved units argument on papiAttributeListAddResolution to the end of the
4587 argument list to match the corresponding get function.
- 4588 • Added papiAttributeListAddCollection and papiAttributeListGetCollection.
- 4589 • Removed unneeded extra level of indirection from attrs argument to
4590 papiAttributeListGet* functions. Also made the attrs argument const.
- 4591 • Added note to "Conventions" section that strings are assumed to be UTF-8
4592 encoded.
- 4593 • Added papiAttributeListFromString and papiAttributeListToString functions,
4594 along with a new appendix defining the string format syntax.
- 4595 • Added papiJobSubmitByReference, papiJobStreamOpen, papiJobStreamWrite,
4596 and papiJobStreamClose functions.
- 4597 • Added short "Document" section in the "Print System Model" chapter.
- 4598 • Added explanation of how multiple files specified in the papiJobSubmit
4599 file_names argument are handled by the print system.
- 4600 • Changed papi_job_ticket_t "uri" field to "file_name" and added explanation text.
- 4601 • Added explanation of implementation option for merging papiJobSubmit
4602 attributes with job_ticket argument.
- 4603 • Added "References" appendix.
- 4604 • Added "IPP Attribute Type Mapping" appendix.
- 4605 • Added "PWG" job ticket format to papi_jt_format_t.
- 4606 • Miscellaneous wording and typo corrections.

4607

4608 **Version 0.5 (August 30, 2002)**

4609

- 4610 • Added job_attrs argument to papiPrinterQuery to support more accurate query
4611 of printer capabilities.
- 4612 • Added management functions papiAttributeDelete, papiJobModify, and
4613 papiPrinterModify.
- 4614 • Added functions papiAttributeListGetValue, papiAttributeListGetString,
4615 papiAttributeListGetInteger, etc.
- 4616 • Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent
4617 with the naming convention (first word after "papi" is the object being operated
4618 upon).
- 4619 • Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.
- 4620 • Made description of authentication more implementation-independent.

- 4621 • Added reference to IPP attributes summary document.
- 4622 • Added result argument to papiPrinterPurgeJobs.
- 4623 • Added "collection attribute" support (PAPI_COLLECTION type).
- 4624 • Changed boolean values to consistently use char. Added PAPI_FALSE and
- 4625 PAPI_TRUE enum values.

4626

4627 **Version 0.4 (July 19, 2002)**

4628

- 4629 • Made papi_job_t and papi_printer_t opaque handles and added "get" functions
- 4630 to access the associated information (papiPrinterGetAttributeList,
- 4631 papiJobGetAttributeList, papiJobGetId, papiJobGetPrinterName,
- 4632 papiJobGetJobTicket).
- 4633 • Removed variable length argument lists from attribute add functions.
- 4634 • Changed order and name of flag value passed to attribute add functions.
- 4635 • Eliminated indirection in date/time value passed to papiAttributeAddDatetime.
- 4636 • Added message argument to papiPrinterPause.

4637

4638 **Version 0.3 (June 24, 2002)**

4639

- 4640 • Converted to DocBook format from Microsoft Word
- 4641 • Major rewrite, including:
 - 4642 • Changed how printer names are described in "Model/Printer"
 - 4643 • Changed fixed length strings to pointers in numerous structures/sections
 - 4644 • Redefined attribute/value structures and associated API descriptions
 - 4645 • Changed list/query functions to return "objects"
 - 4646 • Rewrote "Attributes API" chapter
 - 4647 • Changed many function definitions to pass NULL-terminated arrays of
 - 4648 pointers instead of a separate count argument
 - 4649 • Changed papiJobSubmit to take an attribute list structure as input instead of a
 - 4650 formatted string

4651

4652

4653 **Version 0.2 (April 17, 2002)**

4654

- 4655 • Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)
- 4656 • Filled in "Encryption" section and added information about encryption in "Object
- 4657 Identification" section
- 4658 • Added "short_name" field in "Object Identification" section
- 4659 • Added "Job Ticket (papi_job_ticket_t)" section

Appendix C. Change History

- 4660 • Added papiPrinterPause
- 4661 • Added papiPrinterResume
- 4662 • Added papiPurgeJobs
- 4663 • Added optional job_ticket argument to papiJobSubmit
- 4664 • Added optional passing of filenames by URI to papiJobSubmit
- 4665 • Added papiHoldJob
- 4666 • Added papiReleaseJob
- 4667 • Added papiRestartJob

4668

4669 **Version 0.1 (April 3, 2002)**

4670

- 4671 • Original draft version

4672

4673

4674

4675

4676

4677

<i>End of Document</i>
