

1

2

Open Standard Print API (PAPI)

3

Version 0.8 (V1.0 RELEASE CANDIDATE)

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 0.8 (V1.0 RELEASE CANDIDATE)**

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

14 Version 0.8 (V1.0 RELEASE CANDIDATE) 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
2.4. Globalization	4
3. Common Structures	5
3.1. Conventions.....	5
3.2. Service Object (papi_service_t)	5
3.3. Attributes and Values	5
3.4. Job Object (papi_job_t).....	6
3.5. Stream Object (papi_stream_t).....	6
3.6. Printer Object (papi_printer_t).....	6
3.7. Job Ticket (papi_job_ticket_t).....	7
3.8. Status (papi_status_t).....	7
3.9. List Filter (papi_filter_t).....	8
4. Service API	10
4.1. papiServiceCreate	10
4.2. papiServiceDestroy.....	12
4.3. papiServiceSetUserName	13
4.4. papiServiceSetPassword	14
4.5. papiServiceSetEncryption.....	15
4.6. papiServiceSetAuthCB.....	16
4.7. papiServiceSetAppData	17
4.8. papiServiceGetServiceName	18
4.9. papiServiceGetUserName	19
4.10. papiServiceGetPassword	20
4.11. papiServiceGetEncryption.....	21
4.12. papiServiceGetAppData	22
4.13. papiServiceGetStatusMessage	22
5. Printer API.....	24
5.1. Usage	24
5.2. papiPrintersList.....	24
5.3. papiPrinterQuery.....	26
5.4. papiPrinterModify	28
5.5. papiPrinterPause.....	29
5.6. papiPrinterResume	31
5.7. papiPrinterPurgeJobs	32
5.8. papiPrinterListJobs	34
5.9. papiPrinterGetAttributeList.....	36
5.10. papiPrinterFree	37

67	5.11. papiPrinterListFree.....	38
68	6. Attributes API.....	40
69	6.1. papiAttributeListAdd	40
70	6.2. papiAttributeListAddString.....	41
71	6.3. papiAttributeListAddInteger.....	42
72	6.4. papiAttributeListAddBoolean	43
73	6.5. papiAttributeListAddRange	45
74	6.6. papiAttributeListAddResolution.....	46
75	6.7. papiAttributeListAddDatetime	47
76	6.8. papiAttributeListAddCollection.....	49
77	6.9. papiAttributeDelete.....	50
78	6.10. papiAttributeListGetValue.....	51
79	6.11. papiAttributeListGetString.....	52
80	6.12. papiAttributeListGetInteger.....	54
81	6.13. papiAttributeListGetBoolean.....	55
82	6.14. papiAttributeListGetRange	56
83	6.15. papiAttributeListGetResolution	57
84	6.16. papiAttributeListGetDatetime	59
85	6.17. papiAttributeListGetCollection	60
86	6.18. papiAttributeListFree.....	61
87	6.19. papiAttributeListFind	62
88	6.20. papiAttributeListGetNext.....	63
89	6.21. papiAttributeListFromString	64
90	6.22. papiAttributeListToString	65
91	7. Job API	67
92	7.1. papiJobSubmit.....	67
93	7.2. papiJobSubmitByReference.....	69
94	7.3. papiJobValidate.....	71
95	7.4. papiJobStreamOpen	73
96	7.5. papiJobStreamWrite	75
97	7.6. papiJobStreamClose	76
98	7.7. papiJobQuery	77
99	7.8. papiJobModify	78
100	7.9. papiJobCancel	80
101	7.10. papiJobHold	81
102	7.11. papiJobRelease	83
103	7.12. papiJobRestart	84
104	7.13. papiJobGetAttributeList	85
105	7.14. papiJobGetPrinterName	87
106	7.15. papiJobGetId	88
107	7.16. papiJobGetJobTicket.....	88
108	7.17. papiJobFree.....	89
109	7.18. papiJobListFree	90
110	8. Miscellaneous API	92
111	8.1. papiStatusString.....	92
112	9. Attributes.....	93
113	9.1. Extension Attributes.....	93
114	9.1.1. job-ticket-formats-supported.....	93
115	9.1.2. media-margins.....	93
116	9.2. Required Job Attributes	93
117	9.3. Required Printer Attributes.....	93

118	9.4. IPP Attribute Type Mapping.....	94
119	A. Attribute List Text Representation	95
120	A.1. ABNF Definition.....	95
121	A.2. Examples.....	95
122	B. References	97
123	B.1. Internet Printing Protocol (IPP).....	97
124	97
125	B.2. Job Ticket.....	97
126	97
127	B.3. Printer Working Group (PWG)	97
128	97
129	B.4. Other	97
130	97
131	C. Change History	98

132 Chapter 1. Introduction

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

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

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

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

146 **Chapter 2. Print System Model**

147 **2.1. Introduction**

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

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

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

158

159 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	

160

161

162 **2.2. Model**

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

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

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

174 **2.2.1. Print Service**

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

179 **2.2.2. Printer**

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

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

188 **2.2.3. Job**

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

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

195 **2.2.4. Document**

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

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

203 **2.3. Security**

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

206 **2.3.1. Authentication**

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

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

213 **2.3.2. Authorization**

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

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

222 **2.3.3. Encryption**

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

228 **2.4. Globalization**

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

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

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

239 Chapter 3. Common Structures

240 3.1. Conventions

241

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

247

248 3.2. Service Object (papi_service_t)

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

```
252 typedef void* papi_service_t;
```

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

```
256 typedef enum  
257 {  
258     PAPI_ENCRYPT_IF_REQUESTED, /* Encrypt if requested (TLS upgrade) */  
259     PAPI_ENCRYPT_NEVER,      /* Never encrypt */  
260     PAPI_ENCRYPT_REQUIRED, /* Encryption is required (TLS upgrade) */  
261     PAPI_ENCRYPT_ALWAYS,   /* Always encrypt (SSL) */  
262 } papi_encryption_t;
```

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

266 3.3. Attributes and Values

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

```
269 /* Attribute Type */  
270 typedef enum  
271 {  
272     PAPI_STRING,  
273     PAPI_INTEGER,  
274     PAPI_BOOLEAN,  
275     PAPI_RANGE,  
276     PAPI_RESOLUTION,  
277     PAPI_DATETIME,  
278     PAPI_COLLECTION  
279 } papi_attribute_value_type_t;
```

```
281 /* Resolution units */  
282 typedef enum  
283 {  
284     PAPI_RES_PER_INCH = 3,  
285     PAPI_RES_PER_CM  
286 } papi_res_t;
```

```
288 /* Boolean values */  
289 enum  
290 {  
291     PAPI_FALSE = 0,  
292     PAPI_TRUE = 1  
293 };
```

294

```

295 struct papi_attribute_str;
296
297 /* Attribute Value */
298 typedef union
299 {
300     char* string; /* PAPI_STRING value */
301
302     int integer; /* PAPI_INTEGER value */
303
304     char boolean; /* PAPI_BOOLEAN value */
305
306     struct /* PAPI_RANGE value */
307     {
308         int lower;
309         int upper;
310     } range;
311
312     struct /* PAPI_RESOLUTION value */
313     {
314         int xres;
315         int yres;
316         papi_res_t units;
317     } resolution;
318
319     time_t datetime; /* PAPI_DATETIME value */
320
321     struct papi_attribute_str**
322     collection; /* PAPI_COLLECTION value */
323 } papi_attribute_value_t;
324
325
326 /* Attribute and Values */
327 typedef struct papi_attribute_str
328 {
329     char* name; /* attribute name */
330     papi_attribute_value_type_t type; /* type of values */
331     papi_attribute_value_t** values; /* list of values */
332 } papi_attribute_t;

```

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

```

335 /* Attribute add flags (add_flags) */
336 #define PAPI_ATTR_APPEND 0x0001 /* Add values to attr */
337 #define PAPI_ATTR_REPLACE 0x0002 /* Delete existing
338 values then add new ones */
339 #define PAPI_ATTR_EXCL 0x0004 /* Fail if attr exists */
340

```

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

342 3.4. Job Object (`papi_job_t`)

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

347 3.5. Stream Object (`papi_stream_t`)

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

350 3.6. Printer Object (`papi_printer_t`)

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

355 3.7. Job Ticket (`papi_job_ticket_t`)

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

```

360 /* Job Ticket Format */
361 typedef enum
362 {
363     PAPI_JT_FORMAT_JDF = 0,      /* Job Definition Format */
364     PAPI_JT_FORMAT_PWG = 1     /* PWG Job Ticket Format */
365 } papi_jt_format_t;
366
367
368 /* Job Ticket */
369 typedef struct papi_job_ticket_s
370 {
371     papi_jt_format_t format;      /* Format of job ticket */
372     char* ticket_data;          /* Buffer containing the job
373                                ticket data. If NULL,
374                                file_name must be specified */
375     char* file_name;           /* Name of the file containing
376                                the job ticket data. If
377                                ticket_data is specified, then
378                                file_name is ignored. */
379 } papi_job_ticket_t;
  
```

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

387 3.8. Status (`papi_status_t`)

```

388 typedef enum
389 {
390     PAPI_OK = 0x0000,
391     PAPI_OK_SUBST,
392     PAPI_OK_CONFLICT,
393     PAPI_OK_IGNORED_SUBSCRIPTIONS,
394     PAPI_OK_IGNORED_NOTIFICATIONS,
395     PAPI_OK_TOO_MANY_EVENTS,
396     PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
397     PAPI_REDIRECTION_OTHER_SITE = 0x300,
398     PAPI_BAD_REQUEST = 0x0400,
399     PAPI_FORBIDDEN,
400     PAPI_NOT_AUTHENTICATED,
401     PAPI_NOT_AUTHORIZED,
402     PAPI_NOT_POSSIBLE,
403     PAPI_TIMEOUT,
404     PAPI_NOT_FOUND,
405     PAPI_GONE,
406     PAPI_REQUEST_ENTITY,
407     PAPI_REQUEST_VALUE,
408     PAPI_DOCUMENT_FORMAT,
409     PAPI_ATTRIBUTES,
410     PAPI_URI_SCHEME,
411     PAPI_CHARSET,
412     PAPI_CONFLICT,
413     PAPI_COMPRESSION_NOT_SUPPORTED,
414     PAPI_COMPRESSION_ERROR,
415     PAPI_DOCUMENT_FORMAT_ERROR,
416     PAPI_DOCUMENT_ACCESS_ERROR,
417     PAPI_ATTRIBUTES_NOT_SETTABLE,
418     PAPI_IGNORED_ALL_SUBSCRIPTIONS,
419     PAPI_TOO_MANY_SUBSCRIPTIONS,
420     PAPI_IGNORED_ALL_NOTIFICATIONS,
421     PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
422     PAPI_INTERNAL_ERROR = 0x0500,
423     PAPI_OPERATION_NOT_SUPPORTED,
424     PAPI_SERVICE_UNAVAILABLE,
425     PAPI_VERSION_NOT_SUPPORTED,
426     PAPI_DEVICE_ERROR,
  
```

```

427     PAPI_TEMPORARY_ERROR,
428     PAPI_NOT_ACCEPTING,
429     PAPI_PRINTER_BUSY,
430     PAPI_ERROR_JOB_CANCELLED,
431     PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
432     PAPI_PRINTER_IS_DEACTIVATED,
433     PAPI_BAD_ARGUMENT,
434     PAPI_JOB_TICKET_NOT_SUPPORTED
435 } papi_status_t;
436

```

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

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

442 3.9. List Filter (papi_filter_t)

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

```

446 typedef enum
447 {
448     PAPI_FILTER_BITMASK = 0
449     /* future filter types may be added here */
450 } papi_filter_type_t;
451
452 typedef struct
453 {
454     papi_filter_type_t type; /* Type of filter specified */
455
456     union
457     {
458         /* Bitmask filter */
459         struct
460         {
461             unsigned int mask; /* bit mask */
462             unsigned int value; /* bit value */
463         } bitmask;
464
465         /* future filter types may be added here */
466     } filter;
467 } papi_filter_t;
468

```

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

```

476 papi_filter_t filter;
477 filter.type = PAPI_FILTER_BITMASK;
478 filter.filter.bitmask.mask = PAPI_PRINTER_REMOTE | PAPI_PRINTER_COLOR;
479 filter.filter.bitmask.value = PAPI_PRINTER_COLOR;
480

```

481 The filter bitmask values are:

```

482 enum
483 {
484     PAPI_PRINTER_LOCAL = 0x0000, /* Local printer or class */
485     PAPI_PRINTER_CLASS = 0x0001, /* Printer class */
486     PAPI_PRINTER_REMOTE = 0x0002, /* Remote printer or class */
487     PAPI_PRINTER_BW = 0x0004, /* Can do B&W printing */
488     PAPI_PRINTER_COLOR = 0x0008, /* Can do color printing */
489     PAPI_PRINTER_DUPLEX = 0x0010, /* Can do duplexing */
490     PAPI_PRINTER_STAPLE = 0x0020, /* Can staple output */
491     PAPI_PRINTER_COPIES = 0x0040, /* Can do copies */
492     PAPI_PRINTER_COLLATE = 0x0080, /* Can collage copies */

```

```
493 PAPI_PRINTER_PUNCH = 0x0100, /* Can punch output */
494 PAPI_PRINTER_COVER = 0x0200, /* Can cover output */
495 PAPI_PRINTER_BIND = 0x0400, /* Can bind output */
496 PAPI_PRINTER_SORT = 0x0800, /* Can sort output */
497 PAPI_PRINTER_SMALL = 0x1000, /* Can do Letter/Legal/A4 */
498 PAPI_PRINTER_MEDIUM = 0x2000, /* Can do Tabloid/B/C/A3/A2 */
499 PAPI_PRINTER_LARGE = 0x4000, /* Can do D/E/A1/A0 */
500 PAPI_PRINTER_VARIABLE = 0x8000, /* Can do variable sizes */
501 PAPI_PRINTER_IMPLICIT = 0x10000, /* Implicit class */
502 PAPI_PRINTER_DEFAULT = 0x20000, /* Default printer on network */
503 PAPI_PRINTER_OPTIONS = 0xfffc /* ~(CLASS | REMOTE | IMPLICIT) */
504 };
505
```

506 Chapter 4. Service API

507 4.1. papiServiceCreate

508 Description

509 Create a print service handle to be used in subsequent calls. Memory is allocated
510 and copies of the input arguments are created so that the handle can be used
511 outside the scope of the input variables.

512 The caller must call papiServiceDestroy when done in order to free the resources
513 associated with the print service handle. This must be done even if the
514 papiServiceCreate call failed, because there may be error information associated
515 with the returned handle.

516 Syntax

517

```
518 papi_status_t papiServiceCreate(  
519     papi_service_t* handle,  
520     const char* service_name,  
521     const char* user_name,  
522     const char* password,  
523     const int (*authCB)(papi_service_t svc),  
524     const papi_encryption_t encryption,  
525     void* app_data );  
526
```

527

528 Inputs

529

530 service_name

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

535 user_name

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

539 password

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

542 authCB

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

548 encryption

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

550 app_data

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

553

554 **Outputs**

555

556 handle

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

560

561 **Returns**

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

564 **Example**

565

```

566 #include "papi.h"
567
568 papi_status_t status;
569 papi_service_t handle = NULL;
570 const char* service_name = "ipp:/printserv:631";
571 const char* user_name = "pappy";
572 const char* password = "goober";
573 ...
574 status = papiServiceCreate(&handle,
575                             service_name,
576                             user_name,
577                             password,
578                             NULL,
579                             PAPI_ENCRYPT_IF_REQUESTED,
580                             NULL);
581
582 if (status != PAPI_OK)
583 {
584     /* handle the error */
585     fprintf(stderr, "papiServiceCreate failed: %s\n",
586             papiStatusString(status));
587     if (handle != NULL)
588     {
589         fprintf(stderr, "    details: %s\n",
590                 papiServiceGetStatusMessage(handle));
591     }
592     ...
593 }
594 ...
595 papiServiceDestroy(handle);

```

596

597 **See Also**

598 papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUserName,
599 papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB

600 **4.2. papiServiceDestroy**601 **Description**

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

607 **Syntax**

608

```
609 void papiServiceDestroy(  
610     papi_service_t handle );  
611
```

612

613 **Inputs**

614

615 handle

616 The print service handle to be destroyed.

617

618 **Outputs**

619 none

620 **Returns**

621 none

622 **Example**

623

```
624 #include "papi.h"  
625  
626 papi_status_t status;  
627 papi_service_t handle = NULL;  
628 const char* service_name = "ipp://printserv:631";  
629 const char* user_name = "pappy";  
630 const char* password = "goober";  
631 ...  
632 status = papiServiceCreate(&handle,  
633     service_name,  
634     user_name,  
635     password,  
636     NULL,  
637     PAPI_ENCRYPT_IF_REQUESTED,  
638     NULL);  
639  
640 if (status != PAPI_OK)  
641 {  
642     /* handle the error */  
643     ...  
644 }  
645 ...  
646 papiServiceDestroy(handle);
```

647

648 **See Also**

649 papiServiceCreate

650 4.3. papiServiceSetUserName

651 Description

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

655 Syntax

656

```
657 papi_status_t papiServiceSetUserName(  
658     papi_service_t handle,  
659     const char* user_name );  
660
```

661

662 Inputs

663

664 handle

665 Handle to the print service to update.

666 user_name

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

670

671 Outputs

672 handle is updated.

673

Returns

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

676 Example

677

```
678 #include "papi.h"  
679  
680 papi_status_t status;  
681 papi_service_t handle = NULL;  
682 const char* user_name = "pappy";  
683 ...  
684 status = papiServiceCreate(&handle,  
685     NULL,  
686     NULL,  
687     NULL,  
688     NULL,  
689     PAPI_ENCRYPT_IF_REQUESTED,  
690     NULL);  
691  
692 if (status != PAPI_OK)  
693 {  
694     /* handle the error */  
695     ...  
696 }  
697  
698 status = papiServiceSetUserName(handle, user_name);  
699 if (status != PAPI_OK)  
700 {  
701     /* handle the error */  
702     fprintf(stderr, "papiServiceSetUserName failed: %s\n",  
703         papiServiceGetStatusMessage(handle));
```

```

703     ...
704     }
705     ...
706     papiServiceDestroy(handle);
707

```

708

709 **See Also**

710 papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

711 **4.4. papiServiceSetPassword**712 **Description**

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

716 **Syntax**

717

```

718 papi_status_t papiServiceSetPassword(
719     papi_service_t handle,
720     const char* password );
721

```

722

723 **Inputs**

724

725 handle

726 Handle to the print service to update.

727 password

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

729

730 **Outputs**

731 handle is updated.

732 **Returns**

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

735 **Example**

736

```

737 #include "papi.h"
738
739 papi_status_t status;
740 papi_service_t handle = NULL;
741 const char* password = "goober";
742 ...
743 status = papiServiceCreate(&handle,
744     NULL,
745     NULL,
746     NULL,
747     NULL,
748     PAPI_ENCRYPT_IF_REQUESTED,
749     NULL);
750
751 if (status != PAPI_OK)

```

```

751     {
752         /* handle the error */
753         ...
754     }
755
756     status = papiServiceSetPassword(handle, password);
757     if (status != PAPI_OK)
758     {
759         /* handle the error */
760         fprintf(stderr, "papiServiceSetPassword failed: %s\n",
761             papiServiceGetStatusMessage(handle));
762         ...
763     }
764     ...
765     papiServiceDestroy(handle);
766

```

767

768 **See Also**

769 papiServiceCreate, papiServiceSetUserName, papiServiceGetStatusMessage

770 **4.5. papiServiceSetEncryption**771 **Description**

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

773 **Syntax**

774

```

775 papi_status_t papiServiceSetEncryption(
776     papi_service_t handle,
777     const papi_encryption_t encryption );
778

```

779

780 **Inputs**

781

782 handle

783 Handle to the print service to update.

784 encryption

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

786

787 **Outputs**

788 handle is updated.

789 **Returns**790 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
791 value is returned.792 **Example**

793

```

794 #include "papi.h"
795
796 papi_status_t status;
797 papi_service_t handle = NULL;
798 ...
799 status = papiServiceCreate(&handle,

```

```

800                                     NULL,
801                                     NULL,
802                                     NULL,
803                                     NULL,
804                                     PAPI_ENCRYPT_IF_REQUESTED,
805                                     NULL);
806
807 if (status != PAPI_OK)
808 {
809     /* handle the error */
810     ...
811 }
812
813 status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
814 if (status != PAPI_OK)
815 {
816     /* handle the error */
817     fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
818             papiServiceGetStatusMessage(handle));
819     ...
820 }
821 ...
822 papiServiceDestroy(handle);

```

823

824 **See Also**

825 papiServiceCreate, papiServiceGetStatusMessage

826 **4.6. papiServiceSetAuthCB**827 **Description**

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

830 **Syntax**

831

```

832 papi_status_t papiServiceSetAuthCB(
833     papi_service_t handle,
834     const int (*authCB)(papi_service_t svc) );
835

```

836

837 **Inputs**

838

839 handle

840 Handle to the print service to update.

841 authCB

842 Points to a callback function to be used in authenticating the user to the print
843 service if no password was supplied (or user input is required). A NULL value
844 indicates that no callback should be made. The callback function should return
845 0 if the request is to be cancelled and non-zero if new authentication
846 information has been set.

847

848 **Outputs**

849 handle is updated.

850

Returns

851

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

852

853

Example

854

```

855 #include "papi.h"
856
857 extern int get_password(papi_service_t handle);
858 papi_status_t status;
859 papi_service_t handle = NULL;
860 ...
861 status = papiServiceCreate(&handle,
862                             NULL,
863                             NULL,
864                             NULL,
865                             NULL,
866                             PAPI_ENCRYPT_IF_REQUESTED,
867                             NULL);
868
869 if (status != PAPI_OK)
870 {
871     /* handle the error */
872     ...
873 }
874
875 status = papiServiceSetAuthCB(handle, get_password);
876 if (status != PAPI_OK)
877 {
878     /* handle the error */
879     fprintf(stderr, "papiServiceSetAuthCB failed: %s\n",
880             papiServiceGetStatusMessage(handle));
881     ...
882 }
883 ...
884 papiServiceDestroy(handle);

```

885

886

See Also

887

papiServiceCreate, papiServiceGetStatusMessage

888 4.7. papiServiceSetAppData

889

Description

890

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.

891

892

893

Syntax

894

```

895 papi_status_t papiServiceSetAppData(
896     papi_service_t handle,
897     const void*     app_data );
898

```

899

900

Inputs

901

902 handle

903

Handle to the print service to update.

904 app_data

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

907

908 **Outputs**

909 handle is updated.

910 **Returns**

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

913 **Example**

914

```

915 #include "papi.h"
916
917 extern int get_password(papi_service_t handle);
918 papi_status_t status;
919 papi_service_t handle = NULL;
920 char* app_data = "some data";
921 ...
922 status = papiServiceCreate(&handle,
923                             NULL,
924                             NULL,
925                             NULL,
926                             NULL,
927                             PAPI_ENCRYPT_IF_REQUESTED,
928                             NULL);
929
930 if (status != PAPI_OK)
931 {
932     /* handle the error */
933     ...
934 }
935
936 status = papiServiceSetAppData(handle, app_data);
937 if (status != PAPI_OK)
938 {
939     /* handle the error */
940     fprintf(stderr, "papiServiceSetAppData failed: %s\n",
941             papiServiceGetStatusMessage(handle));
942     ...
943 }
944 ...
945 papiServiceDestroy(handle);

```

946

947 **See Also**

948 papiServiceCreate, papiServiceGetStatusMessage

949 **4.8. papiServiceGetServiceName**

950 **Description**

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

952 **Syntax**

953

```

954 char* papiServiceGetServiceName(
955     papi_service_t handle );
956

```

957

958 **Inputs**

959

960 handle

961 Handle to the print service.

962

963 **Outputs**

964 none

965 **Returns**

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

967 **Example**

968

```

969            #include "papi.h"
970
971            papi_status_t status;
972            papi_service_t handle = NULL;
973            char* service_name = NULL;
974            ...
975            service_name = papiServiceGetServiceName(handle);
976            if (service_name != NULL)
977            {
978                /* use the returned name */
979                ...
980            }
981            ...
982            papiServiceDestroy(handle);
983
```

984

985 **See Also**

986 papiServiceCreate

987 **4.9. papiServiceGetUserName**

988 **Description**

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

990 **Syntax**

991

```

992            char* papiServiceGetUserName(
993                papi_service_t handle );
994
```

995

996 **Inputs**

997

998 handle

999 Handle to the print service.

1000

1001

Outputs

1002

none

1003

Returns

1004

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

1005

Example

1006

1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021

```
#include "papi.h"

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

1022

1023

See Also

1024

papiServiceCreate, papiServiceSetUserName

1025

4.10. papiServiceGetPassword

1026

Description

1027

Get the user password associated with the print service handle.

1028

Syntax

1029

1030
1031
1032

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

1033

1034

Inputs

1035

1036

handle

1037

Handle to the print service.

1038

1039

Outputs

1040

none

1041

Returns

1042

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

1043

Example

1044

```

1045     #include "papi.h"
1046
1047     papi_status_t status;
1048     papi_service_t handle = NULL;
1049     char* password = NULL;
1050     ...
1051     password = papiServiceGetPassword(handle);
1052     if (password != NULL)
1053     {
1054         /* use the returned password */
1055         ...
1056     }
1057     ...
1058     papiServiceDestroy(handle);
1059

```

1060

1061 **See Also**

1062 papiServiceCreate, papiServiceSetPassword

1063 **4.11. papiServiceGetEncryption**1064 **Description**

1065 Get the type of encryption associated with the print service handle.

1066 **Syntax**

1067

```

1068     papi_encryption_t papiServiceGetEncryption(
1069         papi_service_t handle );
1070

```

1071

1072 **Inputs**

1073

1074 handle

1075 Handle to the print service.

1076

1077 **Outputs**

1078 none

1079 **Returns**

1080 The type of encryption associated with the print service handle.

1081 **Example**

1082

```

1083     #include "papi.h"
1084
1085     papi_status_t status;
1086     papi_service_t handle = NULL;
1087     papi_encryption_t encryption;
1088     ...
1089     encryption = papiServiceGetEncryption(handle);
1090     /* use the returned encryption value */
1091     ...
1092     papiServiceDestroy(handle);
1093

```

1094

1095 **See Also**
1096 papiServiceCreate, papiServiceSetEncryption

1097 **4.12. papiServiceGetAppData**

1098 **Description**

1099 Get a pointer to the application-specific data associated with the print service
1100 handle.

1101 **Syntax**

1102

```
1103           void* papiServiceGetAppData(  
1104                    papi_service_t handle );  
1105
```

1106

1107 **Inputs**

1108

1109 handle

1110 Handle to the print service.

1111

1112 **Outputs**

1113 none

1114 **Returns**

1115 A pointer to the application-specific data associated with the print service handle.

1116 **Example**

1117

```
1118           #include "papi.h"  
1119  
1120           papi_status_t status;  
1121           papi_service_t handle = NULL;  
1122           char* app_data = NULL;  
1123           ...  
1124           app_data = (char*)papiServiceGetAppData(handle);  
1125           if (app_data != NULL)  
1126           {  
1127                /* use the returned application data */  
1128                ...  
1129           }  
1130           ...  
1131           papiServiceDestroy(handle);  
1132
```

1133

1134 **See Also**

1135 papiServiceCreate, papiServiceSetAppData

1136 **4.13. papiServiceGetStatusMessage**

1137 **Description**

1138 Get the message associated with the status of the last operation performed. The
1139 status message returned from this function may be more detailed than the status

1140 message returned from `papiStatusString` (if the print service supports returning
1141 more detailed error messages).

1142 The returned message will be localized in the language of the submitter of the
1143 original operation.

1144 **Syntax**

1145

```
1146 const char* papiServiceGetStatusMessage(  
1147     const papi_service_t handle );  
1148
```

1149

1150 **Inputs**

1151

1152 handle

1153 Handle to the print service.

1154

1155 **Outputs**

1156 none

1157 **Returns**

1158 Pointer to the message associated with the status of the last operation performed.

1159 **Example**

1160

```
1161 #include "papi.h"  
1162  
1163 papi_status_t status;  
1164 papi_service_t handle = NULL;  
1165 const char* user_name = "pappy";  
1166 ...  
1167 status = papiServiceCreate(&handle,  
1168                             NULL,  
1169                             NULL,  
1170                             NULL,  
1171                             NULL,  
1172                             PAPI_ENCRYPT_IF_REQUESTED,  
1173                             NULL);  
1174  
1175 if (status != PAPI_OK)  
1176 {  
1177     /* handle the error */  
1178     ...  
1179 }  
1180  
1181 status = papiServiceSetUserName(handle, user_name);  
1182 if (status != PAPI_OK)  
1183 {  
1184     /* handle the error */  
1185     fprintf(stderr, "papiServiceSetUserName failed: %s\n",  
1186             papiServiceGetStatusMessage(handle));  
1187     ...  
1188 }  
1189 ...  
1190 papiServiceDestroy(handle);
```

1191

1192 **See Also**

1193 `papiStatusString`

1194 Chapter 5. Printer API

1195 5.1. Usage

1196 The papiPrinterQuery function queries all/some of the attributes of a printer
1197 object. It returns a list of printer attributes. A successful call to papiPrinterQuery is
1198 typically followed by code which examines and processes the returned attributes.
1199 The using program would then call papiPrinterFree to delete the returned results.

1200 Printers can be found via calls to papiPrintersList. A successful call to
1201 papiPrintersList is typically followed by code to iterate through the list of returned
1202 printers, possibly querying each (papiPrinterQuery) for further information (e.g. to
1203 restrict what printers get displayed for a particular user/request). The using
1204 program would then call papiPrinterListFree to free the returned results.

1205 5.2. papiPrintersList

1206 Description

1207 List all printers known by the print service which match the specified filter.

1208 Depending on the functionality of the target service's "printer directory", the
1209 returned list may be limited to only printers managed by a particular server or it
1210 may include printers managed by other servers.

1211 Syntax

```
1212  
1213 papi_status_t papiPrintersList(  
1214     papi_service_t    handle,  
1215     const char*       requested_attrs[],  
1216     const papi_filter_t* filter,  
1217     papi_printer_t**  printers );  
1218
```

1220 Inputs

1222 handle

1223 Handle to the print service to use.

1224 requested_attrs

1225 (optional) NULL terminated array of attributes to be queried. If NULL is
1226 passed then all attributes are queried. (NOTE: The printer may return more
1227 attributes than you requested. This is merely an advisory request that may
1228 reduce the amount of data returned if the printer/server supports it.)

1229 filter

1230 (optional) Pointer to a filter to limit the number of printers returned on the list
1231 request. See Section 3.9 for details. If NULL is passed then all known printers
1232 are listed.

1233

1234

Outputs

1235

1236 printers

1237

List of printer objects that matched the filter criteria.

1238

1239

Returns

1240

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

1241

1242

Example

1243

1244

```

1245 #include "papi.h"
1246
1247 int i;
1248 papi_status_t status;
1249 papi_service_t handle = NULL;
1250 const char* service_name = "ipp://printserv:631";
1251 const char* user_name = "pappy";
1252 const char* password = "goober";
1253 const char* req_attrs[] =
1254 {
1255     "printer-name",
1256     "printer-location",
1257     NULL
1258 };
1259 papi_filter_t filter;
1260 papi_printer_t* printers = NULL;
1261
1262 /* Select local printers (non-remote) that support color */
1263 filter.type = PAPI_FILTER_BITMASK;
1264 filter.filter.bitmask.mask = PAPI_PRINTER_REMOTE | PAPI_PRINTER_COLOR;
1265 filter.filter.bitmask.value = PAPI_PRINTER_COLOR;
1266 ...
1267 status = papiServiceCreate(&handle,
1268     service_name,
1269     user_name,
1270     password,
1271     NULL,
1272     PAPI_ENCRYPT_IF_REQUESTED,
1273     NULL);
1274
1275 if (status != PAPI_OK)
1276 {
1277     /* handle the error */
1278     ...
1279 }
1280
1281 status = papiPrinterList(handle,
1282     req_attrs,
1283     &filter,
1284     &printers);
1285
1286 if (status != PAPI_OK)
1287 {
1288     /* handle the error */
1289     fprintf(stderr, "papiPrinterList failed: %s\n",
1290         papiServiceGetStatusMessage(handle));
1291     ...
1292 }
1293
1294 if (printers != NULL)
1295 {
1296     for (i=0; printers[i] != NULL; i++)
1297     {
1298         /* process the printer object */
1299         ...
1300     }
1301     papiPrinterListFree(printers);
1302 }
1303
1304 papiServiceDestroy(handle);

```

1303

1304 **See Also**
 1305 papiPrinterListFree, papiPrinterQuery

1306 **5.3. papiPrinterQuery**

1307 **Description**

1308 Queries some or all the attributes of the specified printer object. This includes
 1309 attributes representing the capabilities of the printer, which the caller may use to
 1310 determine which print options to present to the user. How the attributes are
 1311 obtained (e.g. from a static database, from a dialog with the hardware, from a dialog
 1312 with a driver, etc.) is up to the implementer of the API and is beyond the scope of
 1313 this standard.

1314 This optionally includes "context" information which specifies job attributes in the
 1315 context of which the capabilities information is to be constructed.

1316 **Semantics Reference**

1317 Get-Printer-Attributes in [RFC2911], section 3.2.5

1318 **Syntax**

1319

```
1320           papi_status_t papiPrinterQuery(
1321                            papi_service_t     handle,
1322                            const char*        name,
1323                            const char*        requested_attrs[],
1324                            const papi_attribute_t** job_attrs,
1325                            papi_printer_t*    printer );
1326
```

1327

1328 **Inputs**

1329

1330 handle

1331 Handle to the print service to use.

1332 name

1333 The name or URI of the printer to query.

1334 requested_attrs

1335 (optional) NULL terminated array of attributes to be queried. If NULL is
 1336 passed then all attributes are queried. (NOTE: The printer may return more
 1337 attributes than you requested. This is merely an advisory request that may
 1338 reduce the amount of data returned if the printer/server supports it.)

1339 job_attrs

1340 (optional) NULL terminated array of job attributes in the context of which the
 1341 capabilities information is to be constructed. In other words, the returned
 1342 printer attributes represent the capabilities of the printer given that these
 1343 specified job attributes are requested. This allows for more accurate
 1344 information to be retrieved by the caller for a specific job (e.g. "if the job is

1345 printed on A4 size media then duplex output is not available"). If NULL is
 1346 passed then the full capabilities of the printer are queried.

1347 Support for this argument is optional. If the underlying print system does not
 1348 have access to capabilities information bound by job context, then this
 1349 argument may be ignored. But if the calling application will be using the
 1350 returned information to build print job data, then it is always advisable to
 1351 specify the job context attributes. The more context information provided, the
 1352 more accurate capabilities information is likely to be returned from the print
 1353 system.

1354

1355 **Outputs**

1356

1357 printer

Pointer to a printer object containing the requested attributes.

1359

1360 **Returns**

1361 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1362 value is returned.

1363 **Example**

1364

```

1365 #include "papi.h"
1366
1367 papi_status_t status;
1368 papi_service_t handle = NULL;
1369 const char* service_name = "ipp://printserv:631";
1370 const char* user_name = "pappy";
1371 const char* password = "goober";
1372 const char* printer_name = "my-printer";
1373 const char* req_attrs[] =
1374 {
1375     "printer-name",
1376     "printer-location",
1377     "printer-state",
1378     "printer-state-reasons",
1379     "printer-state-message",
1380     NULL
1381 };
1382 papi_attribute_t** job_attrs = NULL;
1383 papi_printer_t printer = NULL;
1384 ...
1385 status = papiServiceCreate(&handle,
1386                             service_name,
1387                             user_name,
1388                             password,
1389                             NULL,
1390                             PAPI_ENCRYPT_IF_REQUESTED,
1391                             NULL);
1392
1393 if (status != PAPI_OK)
1394 {
1395     /* handle the error */
1396     ...
1397 }
1398
1399 papiAttributeListAddString(&job_attrs,
1400                             PAPI_EXCL,
1401                             "media",
1402                             "legal");
1403
1404 status = papiPrinterQuery(handle,
1405                             printer_name,
1406                             req_attrs,
1407                             job_attrs,
1408                             &printer);
1409
1410 if (status != PAPI_OK)
1411 {
  
```

```

1410         /* handle the error */
1411         fprintf(stderr, "papiPrinterQuery failed: %s\n",
1412                papiServiceGetStatusMessage(handle));
1413         ...
1414     }
1415
1416     if (printer != NULL)
1417     {
1418         /* process the printer object */
1419         ...
1420         papiPrinterFree(printer);
1421     }
1422
1423     papiAttributeListFree(job_attrs);
1424     papiServiceDestroy(handle);
1425

```

1426

1427

See Also

1428

papiPrinterList, papiPrinterFree, papiPrinterModify

1429

5.4. papiPrinterModify

1430

Description

1431

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

1432

1433

1434

Semantics Reference

1435

Set-Printer-Attributes in [RFC3380], section 4.1

1436

Syntax

1437

1438

```

1439     papi_status_t papiPrinterModify(
1440         papi_service_t      handle,
1441         const char*         printer_name,
1442         const papi_attribute_t** attrs,
1443         papi_printer_t*     printer );

```

1444

1445

Inputs

1446

1447

handle

1448

Handle to the print service to use.

1449

printer_name

1450

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

1451

attrs

1452

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

1453

1454

Outputs

1455

1456 printer

1457 The modified printer object.

1458

1459 **Returns**

1460 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1461 value is returned.

1462 **Example**

1463

```

1464 #include "papi.h"
1465
1466 papi_status_t status;
1467 papi_service_t handle = NULL;
1468 const char* printer_name = "my-printer";
1469 papi_printer_t printer = NULL;
1470 papi_attribute_t** attrs = NULL;
1471 ...
1472 status = papiServiceCreate(&handle,
1473                            NULL,
1474                            NULL,
1475                            NULL,
1476                            NULL,
1477                            PAPI_ENCRYPT_NEVER,
1478                            NULL);
1479
1480 if (status != PAPI_OK)
1481 {
1482     /* handle the error */
1483     ...
1484 }
1485
1486 papiAttributeListAddString(&attrs,
1487                            PAPI_EXCL,
1488                            "printer-location",
1489                            "Bldg 17/Room 234");
1490
1491 status = papiPrinterModify(handle,
1492                            printer_name,
1493                            attrs,
1494                            &printer);
1495
1496 if (status != PAPI_OK)
1497 {
1498     /* handle the error */
1499     fprintf(stderr, "papiPrinterModify failed: %s\n",
1500            papiServiceGetStatusMessage(handle));
1501     ...
1502 }
1503
1504 if (printer != NULL)
1505 {
1506     /* process the printer */
1507     ...
1508     papiPrinterFree(printer);
1509 }
1510
1511 papiServiceDestroy(handle);

```

1511

1512 **See Also**

1513 papiPrinterQuery, papiPrinterFree

1514 5.5. papiPrinterPause

1515 **Description**

1516 Stops the printer object from scheduling jobs to be printed. Depending on the
1517 implementation, this operation may also stop the printer from processing the
1518 current job(s). This operation is optional and may not be supported by all
1519 printers/servers. Use papiPrinterResume to undo the effects of this operation.

1520 Depending on the implementation, this function may also stop the print service
1521 from processing currently printing job(s).

1522 **Semantics Reference**

1523 Pause-Printer in [RFC2911], section 3.2.7

1524 **Syntax**

1525

```
1526            papi_status_t papiPrinterPause(  
1527                            papi_service_t     handle,  
1528                            const char*        name,  
1529                            const char*        message );  
1530
```

1531

1532 **Inputs**

1533

1534 handle

1535 Handle to the print service to use.

1536 name

1537 The name or URI of the printer to operate on.

1538 message

1539 (optional) An explanatory message to be associated with the paused printer.
1540 This message may be ignored if the underlying print system does not support
1541 associating a message with a paused printer.

1542

1543 **Outputs**

1544 none

1545 **Returns**

1546 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1547 value is returned.

1548 **Example**

1549

```
1550            #include "papi.h"  
1551  
1552            papi_status_t status;  
1553            papi_service_t handle = NULL;  
1554            const char* service_name = "ipp://printserv:631";  
1555            const char* user_name = "pappy";  
1556            const char* password = "goober";  
1557            const char* printer_name = "my-printer";  
1558            ...  
1559            status = papiServiceCreate(&handle,  
1560                                      service_name,  
1561                                      user_name,  
1562                                      password,  
1563                                      NULL,  
1564                                      PAPI_ENCRYPT_IF_REQUESTED,  
1565                                      NULL);  
1566            if (status != PAPI_OK)  
1567            {  
1568                /* handle the error */  
1569                ...  
1570
```

```

1570     }
1571
1572     status = papiPrinterPause(handle, printer_name, NULL);
1573     if (status != PAPI_OK)
1574     {
1575         /* handle the error */
1576         fprintf(stderr, "papiPrinterPause failed: %s\n",
1577                papiServiceGetStatusMessage(handle));
1578         ...
1579     }
1580     ...
1581     papiServiceDestroy(handle);
1582

```

1583

1584 **See Also**

1585 papiPrinterResume

1586 **5.6. papiPrinterResume**1587 **Description**

1588 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the
 1589 effects of papiPrinterPause). This operation is optional and may not be supported
 1590 by all printers/servers, but it must be supported if papiPrinterPause is supported.

1591 **Semantics Reference**

1592 Resume-Printer in [RFC2911], section 3.2.8

1593 **Syntax**

1594

```

1595 papi_status_t papiPrinterResume(
1596             papi_service_t   handle,
1597             const char*      name );
1598

```

1599

1600 **Inputs**

1601

1602 handle

1603 Handle to the print service to use.

1604 name

1605 The name or URI of the printer to operate on.

1606

1607 **Outputs**

1608 none

1609 **Returns**

1610 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1611 value is returned.

1612 **Example**

1613

1614 #include "papi.h"

```

1615
1616     papi_status_t status;
1617     papi_service_t handle = NULL;
1618     const char* service_name = "ipp://printserv:631";
1619     const char* user_name = "pappy";
1620     const char* password = "goober";
1621     const char* printer_name = "my-printer";
1622     ...
1623     status = papiServiceCreate(&handle,
1624                               service_name,
1625                               user_name,
1626                               password,
1627                               NULL,
1628                               PAPI_ENCRYPT_IF_REQUESTED,
1629                               NULL);
1630
1631     if (status != PAPI_OK)
1632     {
1633         /* handle the error */
1634         ...
1635     }
1636
1637     status = papiPrinterPause(handle, printer_name);
1638     if (status != PAPI_OK)
1639     {
1640         /* handle the error */
1641         fprintf(stderr, "papiPrinterPause failed: %s\n",
1642                papiServiceGetStatusMessage(handle));
1643         ...
1644     }
1645     ...
1646     status = papiPrinterResume(handle, printer_name);
1647     if (status != PAPI_OK)
1648     {
1649         /* handle the error */
1650         fprintf(stderr, "papiPrinterResume failed: %s\n",
1651                papiServiceGetStatusMessage(handle));
1652         ...
1653     }
1654     papiServiceDestroy(handle);
1655

```

1656

1657

See Also

1658

papiPrinterPause

1659

5.7. papiPrinterPurgeJobs

1660

Description

1661

Remove all jobs from the specified printer object regardless of their states. This includes removing jobs that have completed and are being kept for history (if any).

1662

This operation is optional and may not be supported by all printers/servers.

1663

1664

Semantics Reference

1665

Purge-Jobs in [RFC2911], section 3.2.9

1666

Syntax

1667

1668

```

papi_status_t papiPrinterPurgeJobs(
1669     papi_service_t handle,
1670     const char* name,
1671     papi_job_t** result);
1672

```

1673

1674

Inputs

1675

1676 handle

1677 Handle to the print service to use.

1678 name

1679 The name or URI of the printer to operate on.

1680

1681 **Outputs**

1682

1683 result

1684 (optional) Pointer to a list of purged jobs with the identifying information (job-
1685 id/job-uri), success/fail, and possibly a detailed message. If NULL is passed
1686 then no job list is returned. Support for the returned job list is optional and may
1687 not be supported by all implementations (if not supported, the function
1688 completes with PAPI_OK_SUBST but no list is returned).

1689 name

1690 The name or URI of the printer to operate on.

1691

1692 **Returns**

1693 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1694 value is returned.

1695 **Example**

1696

```

1697 #include "papi.h"
1698
1699 papi_status_t status;
1700 papi_service_t handle = NULL;
1701 const char* service_name = "ipp://printserv:631";
1702 const char* user_name = "pappy";
1703 const char* password = "goober";
1704 const char* printer_name = "my-printer";
1705 ...
1706 status = papiServiceCreate(&handle,
1707                             service_name,
1708                             user_name,
1709                             password,
1710                             NULL,
1711                             PAPI_ENCRYPT_IF_REQUESTED,
1712                             NULL);
1713
1714 if (status != PAPI_OK)
1715 {
1716     /* handle the error */
1717     ...
1718 }
1719
1720 status = papiPrinterPurgeJobs(handle, printer_name);
1721 if (status != PAPI_OK)
1722 {
1723     /* handle the error */
1724     fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
1725             papiServiceGetStatusMessage(handle));
1726     ...
1727 }
1728
1729 papiServiceDestroy(handle);

```

1730

1731 **See Also**

1732 papiJobCancel

1733 5.8. papiPrinterListJobs

1734 **Description**

1735 List print job(s) associated with the specified printer.

1736 **Semantics Reference**

1737 Get-Jobs in [RFC2911], section 3.2.6

1738 **Syntax**

1739

```

1740 papi_status_t papiPrinterListJobs (
1741             papi_service_t handle,
1742             const char* printer,
1743             const char* requested_attrs[],
1744             const int type_mask,
1745             const int max_num_jobs,
1746             papi_job_t** jobs );
1747
```

1748

1749 **Inputs**

1750

1751 handle

1752 Handle to the print service to use.

1753 requested_attrs

1754 (optional) NULL terminated array of attributes to be queried. If NULL is
 1755 passed then all available attributes are queried. (NOTE: The printer may return
 1756 more attributes than you requested. This is merely an advisory request that
 1757 may reduce the amount of data returned if the printer/server supports it.)

1758 type_mask

1759 A bit mask which determines what jobs will get returned. The following
 1760 constants can be bitwise-OR-ed together to select which types of jobs to list:

```

1761 #define PAPI_LIST_JOBS_OTHERS      0x0001 /* return jobs other than
1762                                           those submitted by the
1763                                           user name assoc with
1764                                           the handle */
1765 #define PAPI_LIST_JOBS_COMPLETED  0x0002 /* return completed jobs */
1766 #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
1767                                           jobs */
1768 #define PAPI_LIST_JOBS_ALL        0xFFFF /* return all jobs */
1769
```

1770

1771 max_num_jobs

1772 Limit to the number of jobs returned. If 0 is passed, then there is no limit on
 1773 the number of jobs which may be returned.

1774

1775

Outputs

1776

1777 jobs

1778

List of job objects returned.

1779

1780

Returns

1781

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

1782

1783

Example

1784

```

1785 #include "papi.h"
1786
1787 int i;
1788 papi_status_t status;
1789 papi_service_t handle = NULL;
1790 const char* printer_name = "my-printer";
1791 papi_job_t* jobs = NULL;
1792 const char* job_attrs[] =
1793 {
1794     "job-id",
1795     "job-name",
1796     "job-originating-user-name",
1797     "job-state",
1798     "job-state-reasons",
1799     NULL
1800 };
1801
1802 ...
1803 status = papiServiceCreate(&handle,
1804                             NULL,
1805                             NULL,
1806                             NULL,
1807                             NULL,
1808                             PAPI_ENCRYPT_NEVER,
1809                             NULL);
1810
1811 if (status != PAPI_OK)
1812 {
1813     /* handle the error */
1814     ...
1815 }
1816
1817 status = papiPrinterListJobs(handle,
1818                             printer_name,
1819                             job_attrs,
1820                             PAPI_LIST_JOBS_ALL,
1821                             0,
1822                             &jobs);
1823
1824 if (status != PAPI_OK)
1825 {
1826     /* handle the error */
1827     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
1828             papiServiceGetStatusMessage(handle));
1829     ...
1830 }
1831
1832 if (jobs != NULL)
1833 {
1834     for(i=0; jobs[i] != NULL; i++)
1835     {
1836         /* process the job */
1837         ...
1838     }
1839     papiJobListFree(jobs);
1840 }
1841
1842 papiServiceDestroy(handle);

```

1841

1842

See Also

1843

papiJobQuery, papiJobListFree

1844 **5.9. papiPrinterGetAttributeList**1845 **Description**

1846 Get the attribute list associated with a printer object.

1847 This function retrieves an attribute list from a printer object returned in a previous
1848 call. Printer objects are returned as the result of operations performed by
1849 papiPrintersList, papiPrinterQuery, and papiPrinterModify.1850 **Syntax**

1851

```
1852 papi_attribute_t** papiPrinterGetAttributeList(
1853     papi_printer_t printer );
1854
```

1855

1856 **Inputs**

1857

1858 printer

Handle of the printer object.

1860

1861 **Outputs**

1862 none

1863 **Returns**

1864 Pointer to the attribute list associated with the printer object.

1865 **Example**

1866

```
1867 #include "papi.h"
1868
1869 papi_status_t status;
1870 papi_service_t handle = NULL;
1871 const char* printer_name = "my-printer";
1872 papi_printer_t printer = NULL;
1873 papi_attribute_list* attrs = NULL;
1874 ...
1875 status = papiServiceCreate(&handle,
1876     NULL,
1877     NULL,
1878     NULL,
1879     NULL,
1880     PAPI_ENCRYPT_NEVER,
1881     NULL);
1882
1883 if (status != PAPI_OK)
1884 {
1885     /* handle the error */
1886     ...
1887 }
1888
1889 status = papiPrinterQuery(handle,
1890     printer_name,
1891     NULL,
1892     &printer);
1893
1894 if (status != PAPI_OK)
1895 {
1896     /* handle the error */
1897     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1898         papiServiceGetStatusMessage(handle));
1899     ...
1900 }
1901
1902 if (printer != NULL)
```

```

1901     {
1902         /* process the printer object */
1903         attrs = papiPrinterGetAttributeList(printer);
1904         ...
1905         papiPrinterFree(printer);
1906     }
1907
1908     papiServiceDestroy(handle);
1909

```

1910

1911 **See Also**

1912 papiPrintersList, papiPrinterQuery

1913 **5.10. papiPrinterFree**1914 **Description**

1915 Free a printer object.

1916 **Syntax**

1917

```

1918 void papiPrinterFree(
1919     papi_printer_t printer );
1920

```

1921

1922 **Inputs**

1923

1924 printer

1925 Handle of the printer object to free.

1926

1927 **Outputs**

1928 none

1929 **Returns**

1930 none

1931 **Example**

1932

```

1933 #include "papi.h"
1934
1935 papi_status_t status;
1936 papi_service_t handle = NULL;
1937 const char* printer_name = "my-printer";
1938 papi_printer_t printer = NULL;
1939 ...
1940 status = papiServiceCreate(&handle,
1941     NULL,
1942     NULL,
1943     NULL,
1944     NULL,
1945     PAPI_ENCRYPT_NEVER,
1946     NULL);
1947
1948 if (status != PAPI_OK)
1949 {
1950     /* handle the error */
1951     ...
1952 }
1953
1954 status = papiPrinterQuery(handle,
1955     printer_name,

```

```

1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973

```

```

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

```

1974

1975 **See Also**

1976 papiPrinterQuery

1977 5.11. papiPrinterListFree

1978 **Description**

1979 Free a list of printer objects.

1980 **Syntax**

1981

```

1982 void papiPrinterListFree(
1983         papi_printer_t*   printers );
1984

```

1985

1986 **Inputs**

1987

1988 printers

1989 Pointer to the printer object list to free.

1990

1991 **Outputs**

1992 none

1993 **Returns**

1994 none

1995 **Example**

1996

```

1997 #include "papi.h"
1998
1999 papi_status_t status;
2000 papi_service_t handle = NULL;
2001 const char* printer_name = "my-printer";
2002 papi_printer_t* printers = NULL;
2003 ...
2004 status = papiServiceCreate(&handle,
2005                             NULL,
2006                             NULL,
2007                             NULL,
2008                             NULL,

```

```

2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037

```

```

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

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

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

papiServiceDestroy(handle);

```

2038

2039

See Also

2040

papiPrinterList

2041 Chapter 6. Attributes API

2042 6.1. papiAttributeListAdd

2043 Description

2044 Add an attribute/value to an attribute list. Depending on the `add_flags`, this may
2045 also be used to add values to an existing multivalued attribute. Memory is allocated
2046 and copies of the input arguments are created. It is the caller's responsibility to call
2047 `papiAttributeListFree` when done with the attribute list.

2048 This function is equivalent to the `papiAttributeListAddString`,
2049 `papiAttributeListAddInteger`, etc. functions defined later in this chapter.

2050 Syntax

2051

```
2052 papi_status_t papiAttributeListAdd(  
2053     papi_attribute_t*** attrs,  
2054     const int add_flags,  
2055     const char* name,  
2056     const papi_attribute_value_type_t type,  
2057     const papi_attribute_value_t* value );  
2058
```

2059

2060 Inputs

2061

2062 `attrs`

2063 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
2064 `NULL` then this function will allocate the attribute list.

2065 `add_flags`

2066 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
2067 that indicates how to handle the request.

2068 `name`

2069 Points to the name of the attribute to add.

2070 `type`

2071 The type of values for this attribute.

2072 `value`

2073 Points to the attribute value to be added.

2074

2075 Outputs

2076

2077 `attrs`

2078 The attribute list is updated.

2079

2080

Returns

2081

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

2082

2083

Example

2084

2085

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

2086

2087

2088

2089

2090

2091

2092

2093

2094

2095

2096

2097

2098

See Also

2099

papiAttributeListFree, papiAttributeListAddString, papiAttributeListAddInteger,

2100

papiAttributeListAddBoolean, papiAttributeListAddRange,

2101

papiAttributeListAddResolution, papiAttributeListAddDatetime

2102

6.2. papiAttributeListAddString

2103

Description

2104

Add a string-valued attribute to an attribute list. Depending on the add_flags, this

2105

may also be used to add values to an existing multivalued attribute. Memory is

2106

allocated and copies of the input arguments are created. It is the caller's

2107

responsibility to call papiAttributeListFree when done with the attribute list.

2108

Syntax

2109

2110

```
papi_status_t papiAttributeListAddString(
```

2111

```
    papi_attribute_t*** attrs,
```

2112

```
    const int add_flags,
```

2113

```
    const char* name,
```

2114

```
    const char* value );
```

2115

2116

Inputs

2117

2118

2119

attrs

2120

Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is

2121

NULL then this function will allocate the attribute list.

2122

add_flags

2123

A mask field consisting of one or more PAPI_ATTR_* values OR-ed together

2124

that indicates how to handle the request.

2125 name
2126 Points to the name of the attribute to add.

2127 value
2128 The value to be added.

2129

2130 **Outputs**

2131

2132 attrs
2133 The attribute list is updated.

2134

2135 **Returns**

2136 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2137 value is returned.

2138 **Example**

2139

```
2140 #include "papi.h"  
2141  
2142 papi_attribute_t** attrs = NULL;  
2143 ...  
2144 papiAttributeListAddString(&attrs,  
2145                             PAPI_EXCL,  
2146                             "job-name",  
2147                             "My job" );  
2148 ...  
2149 papiAttributeListFree(attrs);  
2150
```

2151

2152 **See Also**

2153 papiAttributeListFree, papiAttributeListAdd

2154 **6.3. papiAttributeListAddInteger**

2155 **Description**

2156 Add an integer-valued attribute to an attribute list. Depending on the add_flags,
2157 this may also be used to add values to an existing multivalued attribute. Memory is
2158 allocated and copies of the input arguments are created. It is the caller's
2159 responsibility to call papiAttributeListFree when done with the attribute list.

2160 **Syntax**

2161

```
2162 papi_status_t papiAttributeListAddInteger(  
2163     papi_attribute_t*** attrs,  
2164     const int add_flags,  
2165     const char* name,  
2166     const int value );  
2167
```

2168

2169 **Inputs**

2170

2171 attrs

2172 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
2173 NULL then this function will allocate the attribute list.

2174 add_flags

2175 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
2176 that indicates how to handle the request.

2177 name

2178 Points to the name of the attribute to add.

2179 value

2180 The value to be added.

2181

2182 **Outputs**

2183

2184 attrs

2185 The attribute list is updated.

2186

2187 **Returns**

2188 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2189 value is returned.

2190 **Example**

2191

```
2192           #include "papi.h"
2193
2194           papi_attribute_t** attrs = NULL;
2195           ...
2196           papiAttributeListAddInteger(&attrs,
2197                                        PAPI_EXCL,
2198                                        "copies",
2199                                        3 );
2200           ...
2201           papiAttributeListFree(attrs);
2202
```

2203

2204 **See Also**

2205 papiAttributeListFree, papiAttributeListAdd

2206 **6.4. papiAttributeListAddBoolean**2207 **Description**

2208 Add a boolean-valued attribute to an attribute list. Depending on the add_flags,
2209 this may also be used to add values to an existing multivalued attribute. Memory is
2210 allocated and copies of the input arguments are created. It is the caller's
2211 responsibility to call papiAttributeListFree when done with the attribute list.

2212 **Syntax**

2213

```

2214     papi_status_t papiAttributeListAddBoolean(
2215         papi_attribute_t*** attrs,
2216         const int add_flags,
2217         const char* name,
2218         const char value );
2219 
```

2220

2221 **Inputs**

2222

2223 attrs

2224 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2225 NULL then this function will allocate the attribute list.

2226 add_flags

2227 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2228 that indicates how to handle the request.

2229 name

2230 Points to the name of the attribute to add.

2231 value

2232 The value (PAPI_FALSE or PAPI_TRUE) to be added.

2233

2234 **Outputs**

2235

2236 attrs

2237 The attribute list is updated.

2238

2239 **Returns**

2240 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2241 value is returned.

2242 **Example**

2243

```

2244     #include "papi.h"
2245
2246     papi_attribute_t** attrs = NULL;
2247     ...
2248     papiAttributeListAddBoolean(&attrs,
2249         PAPI_EXCL,
2250         "color-supported",
2251         PAPI_TRUE );
2252     ...
2253     papiAttributeListFree(attrs);
2254 
```

2255

2256 **See Also**
 2257 papiAttributeListFree, papiAttributeListAdd

2258 **6.5. papiAttributeListAddRange**

2259 **Description**

2260 Add a range-valued attribute to an attribute list. Depending on the `add_flags`, this
 2261 may also be used to add values to an existing multivalued attribute. Memory is
 2262 allocated and copies of the input arguments are created. It is the caller's
 2263 responsibility to call `papiAttributeListFree` when done with the attribute list.

2264 **Syntax**

2265

```
2266           papi_status_t papiAttributeListAddRange(  
2267                           papi_attribute_t*** attrs,  
2268                           const int add_flags,  
2269                           const char* name,  
2270                           const int lower,  
2271                           const int upper );  
2272
```

2273

2274 **Inputs**

2275

2276 attrs

2277 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
 2278 `NULL` then this function will allocate the attribute list.

2279 add_flags

2280 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 2281 that indicates how to handle the request.

2282 name

2283 Points to the name of the attribute to add.

2284 lower

2285 The lower range value. This value must be less than or equal to the upper
 2286 range value.

2287 upper

2288 The upper range value. This value must be greater than or equal to the lower
 2289 range value.

2290

2291 **Outputs**

2292

2293 attrs

2294 The attribute list is updated.

2295

2296

Returns

2297

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

2298

2299

Example

2300

2301

```
#include "papi.h"
papi_attribute_t** attrs = NULL;
...
papiAttributeListAddRange (&attrs,
                           PAPI_EXCL,
                           "job-k-octets-supported",
                           1,
                           100000 );
...
papiAttributeListFree (attrs);
```

2302

2303

2304

2305

2306

2307

2308

2309

2310

2311

2312

2313

2314

See Also

2315

papiAttributeListFree

2316

6.6. papiAttributeListAddResolution

2317

Description

2318

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.

2319

2320

2321

2322

Syntax

2323

2324

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

2325

2326

2327

2328

2329

2330

2331

2332

2333

Inputs

2334

2335

attrs

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.

2336

2337

2338

add_flags

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

2339

2340

2341 name
 2342 Points to the name of the attribute to add.
 2343 xres
 2344 The X-axis resolution value.
 2345 yres
 2346 The Y-axis resolution value.
 2347 units
 2348 The units of the resolution values provided.
 2349

2350 **Outputs**

2351
 2352 attrs
 2353 The attribute list is updated.
 2354

2355 **Returns**

2356 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2357 value is returned.

2358 **Example**

```
2359
2360 #include "papi.h"
2361
2362 papi_attribute_t** attrs = NULL;
2363 ...
2364 papiAttributeListAddResolution(&attrs,
2365                               PAPI_EXCL,
2366                               "printer-resolution",
2367                               300,
2368                               300,
2369                               PAPI_RES_PER_INCH );
2370 ...
2371 papiAttributeListFree(attrs);
2372
```

2373

2374 **See Also**

2375 papiAttributeListFree

2376 **6.7. papiAttributeListAddDatetime**

2377 **Description**

2378 Add a date/time-valued attribute to an attribute list. Depending on the add_flags,
 2379 this may also be used to add values to an existing multivalued attribute. Memory is
 2380 allocated and copies of the input arguments are created. It is the caller's
 2381 responsibility to call papiAttributeListFree when done with the attribute list.

2382 **Syntax**

2383

```

2384     papi_status_t papiAttributeListAddDatetime(
2385         papi_attribute_t*** attrs,
2386         const int add_flags,
2387         const char* name,
2388         const time_t date_time );
2389

```

2390

2391 **Inputs**

2392

2393 attrs

2394 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2395 NULL then this function will allocate the attribute list.

2396 add_flags

2397 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2398 that indicates how to handle the request.

2399 name

2400 Points to the name of the attribute to add.

2401 date_time

2402 The date/time value.

2403

2404 **Outputs**

2405

2406 attrs

2407 The attribute list is updated.

2408

2409 **Returns**

2410 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2411 value is returned.

2412 **Example**

2413

```

2414     #include "papi.h"
2415
2416     papi_attribute_t** attrs = NULL;
2417     time_t date_time
2418     ...
2419     time(&date_time);
2420     papiAttributeListAddDatetime(&attrs,
2421         PAPI_EXCL,
2422         "date-time-at-creation",
2423         date_time );
2424     ...
2425     papiAttributeListFree(attrs);
2426

```

2427

2428 **See Also**

2429 `papiAttributeListFree`

2430 **6.8. papiAttributeListAddCollection**

2431 **Description**

2432 Add a collection-valued attribute to an attribute list. Depending on the `add_flags`,
 2433 this may also be used to add values to an existing multivalued attribute. Memory is
 2434 allocated and copies of the input arguments are created. It is the caller's
 2435 responsibility to call `papiAttributeListFree` when done with the attribute list.

2436 **Syntax**

2437

```

2438 papi_status_t papiAttributeListAddCollection(
2439     papi_attribute_t*** attrs,
2440     const int add_flags,
2441     const char* name,
2442     const papi_attribute_t** collection );
2443
```

2444

2445 **Inputs**

2446

2447 `attrs`

2448 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
 2449 `NULL` then this function will allocate the attribute list.

2450 `add_flags`

2451 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 2452 that indicates how to handle the request.

2453 `name`

2454 Points to the name of the attribute to add.

2455 `collection`

2456 The collection value.

2457

2458 **Outputs**

2459

2460 `attrs`

2461 The attribute list is updated.

2462

2463 **Returns**

2464 If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate failure
 2465 value is returned.

2466

Example

2467

2468

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

2469

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

2470

```
papi_attribute_t** collection = NULL;
```

2471

```
...
```

2472

```
/* Build the collection attribute */
```

2473

```
papiAttributeListAddString(&collection,
```

2474

```
    PAPI_EXCL,
```

2475

```
    "media-key",
```

2476

```
    "iso-a4-white");
```

2477

```
papiAttributeListAddString(&collection,
```

2478

```
    PAPI_EXCL,
```

2479

```
    "media-type",
```

2480

```
    "stationery");
```

2481

```
/* Add the collection attribute */
```

2482

```
papiAttributeListAddCollection(&attrs,
```

2483

```
    PAPI_EXCL,
```

2484

```
    "media-col",
```

2485

```
    collection );
```

2486

```
...
```

2487

```
papiAttributeListFree(collection);
```

2488

```
papiAttributeListFree(attrs);
```

2489

2490

2491

2492

2493

See Also

2494

`papiAttributeListFree`

2495

6.9. papiAttributeDelete

2496

Description

2497

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

2498

attribute is freed.

2499

Syntax

2500

2501

```
papi_status_t papiAttributeDelete(
```

2502

```
    papi_attribute_t*** attrs,
```

2503

```
    const char* name);
```

2504

2505

2506

Inputs

2507

2508

`attrs`

2509

Points to an attribute list.

2510

`name`

2511

Points to the name of the attribute to delete.

2512

2513

Outputs

2514

2515 attrs

2516 The attribute list is updated.

2517

2518 **Returns**

2519 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2520 value is returned.

2521 **Example**

2522

```
2523 #include "papi.h"
2524
2525 papi_attribute_t** attrs = NULL;
2526 ...
2527 papiAttributeDelete(&attrs,
2528                   "copies" );
2529 ...
2530
```

2531

2532 **See Also**

2533 papiAttributeListFree

2534 **6.10. papiAttributeListGetValue**

2535 **Description**

2536 Get an attribute's value from an attribute list.

2537 This function is equivalent to the papiAttributeListGetString,
2538 papiAttributeListGetInteger, etc. functions defined later in this chapter.

2539 **Syntax**

2540

```
2541 papi_status_t papiAttributeListGetValue(
2542     const papi_attribute_t** attrs,
2543     void** iterator,
2544     const char* name,
2545     const papi_attribute_value_type_t type,
2546     papi_attribute_value_t** 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 type
 2562 The type of values for this attribute.

2563

2564 **Outputs**

2565

2566 value
 2567 Points to the variable where a pointer to the attribute value is to be returned.
 2568 Note that the returned pointer points to the attribute's value in the list (no copy
 2569 of the value is made) so that the caller does not need to do any special cleanup
 2570 of the returned value's memory (it is cleaned up when the containing attribute
 2571 list is freed).

2572 If this call returns an error, the output value is not changed.

2573

2574 **Returns**

2575 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2576 value is returned.

2577 **Example**

2578

```

2579 #include "papi.h"
2580
2581 papi_attribute_t** attrs = NULL;
2582 papi_attribute_value_t* job_name_value;
2583 ...
2584 papiAttributeListGetValue(attrs,
2585     NULL,
2586     "job-name",
2587     PAPI_STRING,
2588     &job_name_value );
2589 if (job_name_value != NULL)
2590 {
2591     /* process the value */
2592     ...
2593 }
2594 ...
2595 papiAttributeListFree(attrs);
2596
    
```

2597

2598 **See Also**

2599 papiAttributeListFree, papiAttributeListGetString, papiAttributeListGetInteger,
 2600 papiAttributeListGetBoolean, papiAttributeListGetRange,
 2601 papiAttributeListGetResolution, papiAttributeListGetDatetime

2602 **6.11. papiAttributeListGetString**

2603 **Description**

2604 Get a string-valued attribute's value from an attribute list.

2605 **Syntax**

2606

```

2607     papi_status_t papiAttributeListGetString(
2608         const papi_attribute_t** attrs,
2609         void** iterator,
2610         const char* name,
2611         char** value );
2612

```

2613

2614 **Inputs**

2615

2616 attrs

2617 The attribute list.

2618 iterator

2619 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2620 then only the first value is returned, even if the attribute is multivalued. If the
 2621 argument points to a void* that is set to NULL, then the first attribute value is
 2622 returned and the iterator can then be passed in unchanged on subsequent calls
 2623 to this function to get the remaining values.

2624 name

2625 Points to the name of the attribute whose value to get.

2626

2627 **Outputs**

2628

2629 value

2630 Pointer to the char* where a pointer to the value is returned. If this call returns
 2631 an error, the output value is not changed.

2632

2633 **Returns**

2634 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2635 value is returned.

2636 **Example**

2637

```

2638 #include "papi.h"
2639
2640 papi_attribute_t** attrs = NULL;
2641 char* job_name_value = NULL;
2642 ...
2643 papiAttributeListGetString(attrs,
2644     NULL,
2645     "job-name",
2646     &job_name_value );
2647
2648 if (job_name_value != NULL)
2649 {
2650     /* process the value */
2651     ...
2652 }
2653 ...
2654 papiAttributeListFree(attrs);

```

2655

2656

See Also

2657

papiAttributeListFree, papiAttributeListGetValue

2658

6.12. papiAttributeListGetInteger

2659

Description

2660

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

2661

Syntax

2662

2663

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

2664

2665

2666

2667

2668

2669

2670

Inputs

2671

2672

attrs

2673

The attribute list.

2674

iterator

2675

(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.

2676

2677

2678

2679

2680

name

2681

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

2682

2683

Outputs

2684

2685

value

2686

Pointer to the int where the value is returned. If this call returns an error, the output value is not changed.

2687

2688

2689

Returns

2690

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

2691

2692

Example

2693

2694

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

2695

```

2696     papi_attribute_t** attrs = NULL;
2697     int copies = 0;
2698     ...
2699     papiAttributeListGetInteger(attrs,
2700                               NULL,
2701                               "copies",
2702                               &copies );
2703     /* process the value */
2704     ...
2705     papiAttributeListFree(attrs);
2706

```

2707

2708 **See Also**

2709 papiAttributeListFree, papiAttributeListGetValue

2710 **6.13. papiAttributeListGetBoolean**2711 **Description**

2712 Get an boolean-valued attribute's value from an attribute list.

2713 **Syntax**

2714

```

2715     papi_status_t papiAttributeListGetBoolean(
2716         const papi_attribute_t** attrs,
2717         void** iterator,
2718         const char* name,
2719         char* value );
2720

```

2721

2722 **Inputs**

2723

2724 attrs

2725 The attribute list.

2726 iterator

2727 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2728 then only the first value is returned, even if the attribute is multivalued. If the
 2729 argument points to a void* that is set to NULL, then the first attribute value is
 2730 returned and the iterator can then be passed in unchanged on subsequent calls
 2731 to this function to get the remaining values.

2732 name

2733 Points to the name of the attribute whose value to get.

2734

2735 **Outputs**

2736

2737 value

2738 Pointer to the char where the value is returned. If this call returns an error, the
 2739 output value is not changed.

2740

2741

Returns

2742

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

2743

2744

Example

2745

2746

```
#include "papi.h"
papi_attribute_t** attrs = NULL;
char_color_supp = PAPI_FALSE;
...
papiAttributeListGetBoolean(attrs,
                            NULL,
                            "color-supported",
                            &color_supp );
/* process the value */
...
papiAttributeListFree(attrs);
```

2747

2748

2749

2750

2751

2752

2753

2754

2755

2756

2757

2758

2759

2760

See Also

2761

papiAttributeListFree, papiAttributeListGetValue

2762

6.14. papiAttributeListGetRange

2763

Description

2764

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

2765

Syntax

2766

2767

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

2768

2769

2770

2771

2772

2773

2774

2775

Inputs

2776

2777

attrs

2778

The attribute list.

2779

iterator

2780

(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.

2781

2782

2783

2784

2785

name

2786

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

2787

2788

Outputs

2789

2790 lower

2791

Pointer to the int where the lower range value is returned. If this call returns an error, the output value is not changed.

2792

2793 upper

2794

Pointer to the int where the upper range value is returned. If this call returns an error, the output value is not changed.

2795

2796

2797

Returns

2798

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

2799

2800

Example

2801

2802

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
int lower = 0;
int upper = 0;
...
papiAttributeListGetRange(attrs,
    NULL,
    "job-k-octets-supported",
    &lower,
    &upper );
/* process the value */
...
papiAttributeListFree(attrs);
```

2803

2804

2805

2806

2807

2808

2809

2810

2811

2812

2813

2814

2815

2816

2817

2818

See Also

2819

papiAttributeListFree, papiAttributeListGetValue

2820

6.15. papiAttributeListGetResolution

2821

Description

2822

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

2823

Syntax

2824

2825

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

2826

2827

2828

2829

2830

2831

2832

2833

2834 **Inputs**

2835

2836 attrs

2837 The attribute list.

2838 iterator

2839 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2840 then only the first value is returned, even if the attribute is multivalued. If the
 2841 argument points to a void* that is set to NULL, then the first attribute value is
 2842 returned and the iterator can then be passed in unchanged on subsequent calls
 2843 to this function to get the remaining values.

2844 name

2845 Points to the name of the attribute whose value to get.

2846

2847 **Outputs**

2848

2849 xres

2850 Pointer to the int where the X-resolution value is returned. If this call returns
 2851 an error, the output value is not changed.

2852 yres

2853 Pointer to the int where the Y-resolution value is returned. If this call returns
 2854 an error, the output value is not changed.

2855 units

2856 Pointer to the variable where the resolution-units value is returned. If this call
 2857 returns an error, the output value is not changed.

2858

2859 **Returns**

2860 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2861 value is returned.

2862 **Example**

2863

```

2864 #include "papi.h"
2865
2866 papi_attribute_t** attrs = NULL;
2867 int xres = 0;
2868 int yres = 0;
2869 papi_res_t units;
2870 ...
2871 papiAttributeListGetResolution(attrs,
2872                               NULL,
2873                               "printer-resolution",
2874                               &xres,
2875                               &yres,
2876                               &units );
2877 /* process the value */
2878 ...
2879 papiAttributeListFree(attrs);
2880
    
```


2881

2882

See Also

2883

papiAttributeListFree, papiAttributeListGetValue

2884

6.16. papiAttributeListGetDatetime

2885

Description

2886

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

2887

Syntax

2888

2889

```
papi_status_t papiAttributeListGetDatetime(
2890     const papi_attribute_t** attrs,
2891     void** iterator,
2892     const char* name,
2893     time_t* date_time );
2894
```

2895

2896

Inputs

2897

2898 attrs

2899

The attribute list.

2900 iterator

2901

(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.

2902

2903

2904

2905

2906 name

2907

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

2908

2909

Outputs

2910

2911 date_time

2912

Pointer to the variable where the date/time value is returned. If this call returns an error, the output value is not changed.

2913

2914

2915

Returns

2916

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

2917

2918

Example

2919

```

2920     #include "papi.h"
2921
2922     papi_attribute_t** attrs = NULL;
2923     time_t date_time;
2924     ...
2925     papiAttributeListGetDatetime(attrs,
2926                                 NULL,
2927                                 "date-time-at-creation",
2928                                 &date_time );
2929     /* process the value */
2930     ...
2931     papiAttributeListFree(attrs);
2932

```

2933

2934 **See Also**

2935 papiAttributeListFree, papiAttributeListGetValue

2936 **6.17. papiAttributeListGetCollection**2937 **Description**

2938 Get a collection-valued attribute's value from an attribute list.

2939 **Syntax**

2940

```

2941     papi_status_t papiAttributeListGetCollection(
2942         const papi_attribute_t** attrs,
2943         void** iterator,
2944         const char* name,
2945         papi_attribute_t*** collection );
2946

```

2947

2948 **Inputs**

2949

2950 attrs

2951 The attribute list.

2952 iterator

2953 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2954 then only the first value is returned, even if the attribute is multivalued. If the
 2955 argument points to a void* that is set to NULL, then the first attribute value is
 2956 returned and the iterator can then be passed in unchanged on subsequent calls
 2957 to this function to get the remaining values.

2958 name

2959 Points to the name of the attribute whose value to get.

2960

2961 **Outputs**

2962

2963 collection

2964 Pointer to the attribute list where a pointer to the collection value is returned.
 2965 Note that the value is not copied, so the caller does not need to free the
 2966 returned list (it will be freed when the containing attribute list is freed).

2967 If this call returns an error, the output value is not changed.

2968

2969 **Returns**

2970 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2971 value is returned.

2972 **Example**

2973

```

2974 #include "papi.h"
2975
2976 papi_attribute_t** attrs = NULL;
2977 papi_attribute_t** collection = NULL;
2978 ...
2979 papiAttributeListGetCollection(attrs,
2980                               NULL,
2981                               "media-col",
2982                               &collection );
2983 /* process the value */
2984 ...
2985 papiAttributeListFree(attrs);
2986

```

2987

2988 **See Also**

2989 papiAttributeListFree, papiAttributeListGetValue

2990 **6.18. papiAttributeListFree**

2991 **Description**

2992 Frees an attribute list.

2993 **Syntax**

2994

```

2995 void papiAttributeListFree(
2996     const papi_attribute_t** attrs );
2997

```

2998

2999 **Inputs**

3000

3001 attrs

3002 Attribute list to be freed.

3003

3004 **Outputs**

3005 none

3006 **Returns**

3007 none

3008 **Example**

3009

```

3010 #include "papi.h"
3011
3012 papi_attribute_t** attrs = NULL;
3013 ...
3014 papiAttributeListAddString(&attrs,
3015                             "job-name",
3016                             PAPI_EXCL,
3017                             1,
3018                             "My job" );
3019 ...
3020 papiAttributeListFree(attrs);
3021

```

3022

3023 **See Also**

3024 papiAttributeListAddString, etc.

3025 6.19. papiAttributeListFind

3026 **Description**

3027 Find an attribute in an attribute list.

3028 **Syntax**

3029

```

3030 papi_attribute_t* papiAttributeListFind(
3031     const papi_attribute_t** attrs,
3032     const char* name );
3033

```

3034

3035 **Inputs**

3036

3037 attrs

Attribute list to be searched.

3038

3039 name

Pointer to the name of the attribute to find.

3040

3041

3042 **Outputs**

3043 none

3044 **Returns**

3045 Pointer to the found attribute. NULL indicates that the specified attribute was not
 3046 found

3047 **Example**

3048

```

3049     #include "papi.h"
3050
3051     papi_attribute_t** attrs = NULL;
3052     papi_attribute_t* attr = NULL;
3053     ...
3054     attr = papiAttributeListFind(&attrs,
3055                                 "job-name" );
3056     if (attr != NULL)
3057     {
3058         /* process the attribute */
3059         ...
3060     }
3061     ...
3062     papiAttributeListFree(attrs);
3063

```

3064

3065 **See Also**

3066 papiAttributeListGetNext

3067 **6.20. papiAttributeListGetNext**3068 **Description**

3069 Get the next attribute in an attribute list.

3070 **Syntax**

3071

```

3072     papi_attribute_t* papiAttributeListGetNext(
3073         const papi_attribute_t** attrs,
3074         void** iterator );
3075

```

3076

3077 **Inputs**

3078

3079 attrs

3080 Attribute list to be used.

3081 iterator

3082 Pointer to an opaque (void*) iterator. This should be NULL to find the first
 3083 attribute and then passed in unchanged on subsequent calls to this function.

3084

3085 **Outputs**

3086 none

3087 **Returns**

3088 Pointer to the found attribute. NULL indicates that the end of the attribute list was
 3089 reached.

3090 **Example**

3091

```

3092     #include "papi.h"
3093
3094     papi_attribute_t** attrs = NULL;
3095     papi_attribute_t* attr = NULL;
3096     void* iterator = NULL;

```

```

3097     ...
3098     attr = papiAttributeListGetNext (&attrs,
3099                                     &iterator );
3100
3101     while (attr != NULL)
3102     {
3103         /* process this attribute */
3104         ...
3105         attr = papiAttributeListGetNext (&attrs,
3106                                         &iterator );
3107     }
3108     ...
3109     papiAttributeListFree (attrs);

```

3110

3111

See Also

3112

papiAttributeListFind

3113 **6.21. papiAttributeListFromString**

3114

Description

3115

Convert a string of text options to an attribute list.

3116

3117

3118

3119

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

3120

Syntax

3121

```

3122     papi_status_t papiAttributeListFromString(
3123         papi_attribute_t*** attrs,
3124         const int add_flags,
3125         const char* buffer );
3126

```

3127

3128

Inputs

3129

3130 attrs

3131

3132

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.

3133 add_flags

3134

3135

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

3136 buffer

3137

3138

Points to text options.

3139

Outputs

3140

3141 attrs

3142 The attribute list is updated.

3143

3144 **Returns**

3145 If the text string is successfully converted to an attribute list, a value of PAPI_OK is
3146 returned. Otherwise an appropriate failure value is returned.

3147 **Example**

3148

```
3149 #include "papi.h"
3150
3151 papi_attribute_t** attrs = NULL;
3152 char buffer[8192];
3153 ...
3154 strcpy(buffer,
3155 "copies=1 job-name=John's\ Really\040Nice\ Job");
3156
3157 papiAttributeListFromString(&attrs, PAPI_EXCL, buffer);
3158 ...
3159 papiAttributeListFree(attrs);
3160
```

3161

3162 **See Also**

3163 papiAttributeListToString

3164 6.22. papiAttributeListToString

3165 **Description**

3166 Convert an attribute list to its text representation. The destination string is limited
3167 to at most (buflen - 1) bytes plus the trailing nul byte.

3168 PAPI provides two functions which map job attributes to and from text options
3169 that are typically provided on the command-line by the user. This text encoding is
3170 also backwards-compatible with existing printing systems and is relatively simple
3171 to parse and generate. See Appendix A for a definition of the string syntax.

3172 **Syntax**

3173

```
3174 papi_status_t papiAttributeListToString(
3175     const papi_attribute_t** attrs,
3176     const char* attr_delim,
3177     char* buffer,
3178     const size_t buflen );
3179
```

3180

3181 **Inputs**

3182

3183 attrs

3184 Points to an attribute list.

3185 attr_delim
3186 (optional) If not NULL, points to a string to be placed between attributes in the
3187 output buffer. If NULL, a space is used as the attribute delimiter.

3188 buffer
3189 Points to a string buffer to receive the to receive the text representation of the
3190 attribute list.

3191 buflen
3192 Specifies the length of the string buffer in bytes.
3193

3194 **Outputs**

3195
3196 buffer
3197 The buffer is filled with the text representation of the attribute list. The buffer
3198 will always be set to something by this function (buffer[0] = NULL in cases of
3199 an error).

3200

3201 **Returns**

3202 If the attribute list is successfully converted to a text string, a value of PAPI_OK is
3203 returned. Otherwise an appropriate failure value is returned.

3204 **Example**

3205

```
3206 #include "papi.h"  
3207  
3208 papi_attribute_t** attrs = NULL;  
3209 char buffer[8192];  
3210 ...  
3211 papiAttributeListToString(attrs, NULL, buffer, sizeof(buffer));  
3212 ...  
3213 papiAttributeListFree(attrs);  
3214
```

3215

3216 **See Also**

3217 papiAttributeListFromString

3218 **Chapter 7. Job API**

3219 **7.1. papiJobSubmit**

3220 **Description**

3221 Submits a print job having the specified attributes to the specified printer. This
3222 interface copies the specified print files before returning to the caller (contrast to
3223 papiJobSubmitByReference). The caller must call papiJobFree when done in order to
3224 free the resources associated with the returned job object.

3225 Attributes of the print job may be passed in the job_attributes argument and/or in
3226 a job ticket (using the job_ticket argument). If both are specified, the attributes in the
3227 job_attributes list will be applied to the job_ticket attributes and the resulting
3228 attribute set will be used.

3229 **Semantics Reference**

3230 Print-Job in [RFC2911], section 3.2.1

3231 **Syntax**

3232

```
3233 papi_status_t papiJobSubmit(  
3234     papi_service_t      handle,  
3235     const char*         printer_name,  
3236     const papi_attribute_t** job_attributes,  
3237     const papi_job_ticket_t* job_ticket,  
3238     const char**        file_names,  
3239     papi_job_t*         job );  
3240
```

3241

3242 **Inputs**

3243

3244 handle

3245 Handle to the print service to use.

3246 printer_name

3247 Pointer to the name of the printer to which the job is to be submitted.

3248 job_attributes

3249 (optional) The list of attributes describing the job and how it is to be printed. If
3250 options are specified here and also in the job ticket data, the value specified
3251 here takes precedence. If this is NULL then only default attributes and
3252 (optionally) a job ticket is submitted with the job.

3253 job_ticket

3254 (optional) Pointer to structure specifying the job ticket. If this argument is
3255 NULL, then no job ticket is used with the job.

3256 Whether the implementation passes both the attributes and the job ticket to the
3257 server/printer, or merges them to some print protocol or internal
3258 implementation depends on the implementation.

3259 file_names

3260 NULL terminated list of pointers to names of files to print. If more than one
 3261 file is specified, the files will be treated by the print system as separate
 3262 "documents" for things like page breaks and separator sheets, but they will be
 3263 scheduled and printed together as one job and the specified attributes will
 3264 apply to all the files.

3265 These file names may contain absolute path names or relative path names
 3266 (relative to the current path). The implementation MUST copy the file contents
 3267 before returning.

3268

3269 **Outputs**

3270

3271 job

The resulting job object representing the submitted job.

3273

3274 **Returns**

3275 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3276 value is returned.

3277 **Example**

3278

```

3279 #include "papi.h"
3280
3281 papi_status_t status;
3282 papi_service_t handle = NULL;
3283 const char* printer = "my-printer";
3284 const papi_attribute_t** attrs = NULL;
3285 const papi_job_ticket_t* ticket = NULL;
3286 const char* files[] = { "/etc/motd", NULL };
3287 papi_job_t job = NULL;
3288
3289 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3290                           PAPI_ENCRYPT_IF_REQUESTED, NULL);
3291 if (status != PAPI_OK)
3292 {
3293     /* handle the error */
3294     ...
3295 }
3296
3297 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3298                            PAPI_STRING, 1, "test job");
3299 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3300                             PAPI_INTEGER, 1, 4);
3301
3302 status = papiJobSubmit(handle,
3303                       printer,
3304                       attrs,
3305                       ticket,
3306                       files,
3307                       &job);
3308 if (status != PAPI_OK)
3309 {
3310     fprintf(stderr, "papiJobSubmit failed: %s\n",
3311            papiStatusString(status));
3312     ...
3313 }
3314
3315 if (job != NULL)
3316 {
3317     /* look at the job object (maybe get the id) */
3318     papiJobFree(job);
3319 }
3320
3321 papiServiceDestroy(handle);
3322
3323 
```

3324

3325

See Also

3326

papiJobSubmitByReference, papiJobValidate, papiJobFree

3327

7.2. papiJobSubmitByReference

3328

Description

3329

3330

3331

3332

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

3333

3334

3335

3336

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

3337

Semantics Reference

3338

Print-URI in [RFC2911], section 3.2.2

3339

Syntax

3340

3341

3342

3343

3344

3345

3346

3347

3348

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

3349

3350

Inputs

3351

3352

handle

3353

Handle to the print service to use.

3354

printer_name

3355

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

3356

job_attributes

3357

3358

3359

3360

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

3361

job_ticket

3362

3363

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

3364 Whether the implementation passes both the attributes and the job ticket to the
 3365 server/printer, or merges them to some print protocol or internal
 3366 implementation depends on the implementation.

3367 file_names

3368 NULL terminated list of pointers to names of files to print. If more than one
 3369 file is specified, the files will be treated by the print system as separate
 3370 "documents" for things like page breaks and separator sheets, but they will be
 3371 scheduled and printed together as one job and the specified attributes will
 3372 apply to all the files.

3373 These file names may contain absolute path names, relative path names or
 3374 URIs ([RFC1738], [RFC2396]). The implementation SHOULD NOT copy the
 3375 referenced data unless (or until) it is no longer feasible to maintain the
 3376 reference. Feasibility limitations may arise out of security issues, namespace
 3377 issues, and/or protocol or printer limitations.

3378 Implementations MUST support the absolute path, relative path, and "file:"
 3379 URI scheme. Use of other URI schemes could result in a PAPI_URI_SCHEME
 3380 error, depending on the implementation.

3381 The semantics explained in the preceding paragraphs allows for flexibility in
 3382 the PAPI implementation. For example: (1) PAPI on top of a local service to
 3383 maintain the reference for the life of the job, if the local service supports it. (2)
 3384 PAPI on top of IPP to send a reference when the server can access the
 3385 referenced data and copy it when it is not accessible to the server. (3) PAPI on
 3386 top of network printing protocols that don't support references to copy the data
 3387 on the way out to the remote server.

3388

3389 **Outputs**

3390

3391 job

3392 The resulting job object representing the submitted job.

3393

3394 **Returns**

3395 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3396 value is returned.

3397 **Example**

3398

```

3399 #include "papi.h"
3400
3401 papi_status_t status;
3402 papi_service_t handle = NULL;
3403 const char* printer = "my-printer";
3404 const papi_attribute_t** attrs = NULL;
3405 const papi_job_ticket_t* ticket = NULL;
3406 const char* files[] = { "http://foo.bar.org/docs/glop.pdf", NULL };
3407 papi_job_t job = NULL;
3408
3409 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3410                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
3411 if (status != PAPI_OK)
3412 {
3413     /* handle the error */
3414     ...
3415 }

```

```

3416
3417     papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3418                               PAPI_STRING, 1, "test job");
3419
3420     papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3421                                PAPI_INTEGER, 1, 4);
3422
3423     status = papiJobSubmitByReference(handle,
3424                                       printer,
3425                                       attrs,
3426                                       ticket,
3427                                       files,
3428                                       &job);
3429
3430     if (status != PAPI_OK)
3431     {
3432         fprintf(stderr, "papiJobSubmitByReference failed: %s\n",
3433                papiStatusString(status));
3434         ...
3435     }
3436
3437     if (job != NULL)
3438     {
3439         /* look at the job object (maybe get the id) */
3440         papiJobFree(job);
3441     }
3442
3443     papiServiceDestroy(handle);

```

3444

3445

See Also

3446

papiJobSubmit, papiJobValidate, papiJobFree

3447

7.3. papiJobValidate

3448

Description

3449

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

3450

3451

3452

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

3453

3454

3455

3456

Semantics Reference

3457

Validate-Job in [RFC2911], section 3.2.3

3458

Syntax

3459

3460

3461

3462

3463

3464

3465

3466

3467

```

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

```

3468

3469

Inputs

3470

3471 handle
 3472 Handle to the print service to use.

3473 printer_name
 3474 Pointer to the name of the printer against which the job is to be validated.

3475 job_attributes
 3476 (optional) The list of attributes describing the job and how it is to be printed. If
 3477 options are specified here and also in the job ticket data, the value specified
 3478 here takes precedence. If this is NULL then only default attributes and
 3479 (optionally) a job ticket is submitted with the job.

3480 job_ticket
 3481 (optional) Pointer to structure specifying the JDF job ticket. If this argument is
 3482 NULL, then no job ticket is used with the job.

3483 file_names
 3484 NULL terminated list of pointers to names of files to validate.

3486 **Outputs**

3488 job
 3489 The resulting job object representing what would be the submitted job.

3491 **Returns**

3492 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3493 value is returned.

3494 **Example**

```

3496 #include "papi.h"
3497
3498 papi_status_t status;
3499 papi_service_t handle = NULL;
3500 const char* printer = "my-printer";
3501 const papi_attribute_t** attrs = NULL;
3502 const papi_job_ticket_t* ticket = NULL;
3503 const char* files[] = { "/etc/motd", NULL };
3504 papi_job_t job = NULL;
3505
3506 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3507                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
3508 if (status != PAPI_OK)
3509 {
3510     /* handle the error */
3511     ...
3512 }
3513
3514 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3515                          PAPI_STRING, 1, "test job");
3516 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3517                            PAPI_INTEGER, 1, 4);
3518
3519 status = papiJobValidate(handle,
3520                        printer,
3521                        attrs,
3522                        ticket,
3523                        files,
3524                        &job);
    
```

```

3525     if (status != PAPI_OK)
3526     {
3527         fprintf(stderr, "papiJobValidate failed: %s\n",
3528                papiStatusString(status));
3529         ...
3530     }
3531
3532     if (job != NULL)
3533     {
3534         ...
3535         papiJobFree(job);
3536     }
3537
3538     papiServiceDestroy(handle);
3539

```

3540

3541

See Also

3542

papiJobSubmit, papiJobFree

3543

7.4. papiJobStreamOpen

3544

Description

3545

Opens a print job and an associated stream of print data to be sent to the specified printer. After calling this function papiJobStreamWrite can be called (repeatedly) to write the print data to the stream, and then papiJobStreamClose is called to complete the submission of the print job.

3546

3547

3548

3549

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

3550

3551

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

3552

3553

3554

3555

Syntax

3556

3557

```

papi_status_t papiJobStreamOpen(
3558     papi_service_t      handle,
3559     const char*         printer_name,
3560     const papi_attribute_t** job_attributes,
3561     const papi_job_ticket_t* job_ticket,
3562     papi_stream_t*      stream );
3563

```

3564

3565

Inputs

3566

3567

handle

3568

Handle to the print service to use.

3569

printer_name

3570

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

3571 job_attributes
 3572 (optional) The list of attributes describing the job and how it is to be printed.
 3573 See job_attributes argument for papiJobSubmit for description.

3574 job_ticket
 3575 (optional) Pointer to structure specifying the job ticket. See job_ticket argument
 3576 for papiJobSubmit for description.

3577

3578 **Outputs**

3579

3580 stream

3581 The resulting stream object to which print data can be written.

3582

3583 **Returns**3584 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3585 value is returned.3586 **Example**

3587

```

3588 #include "papi.h"
3589
3590 papi_status_t status;
3591 papi_service_t handle = NULL;
3592 const char* printer = "my-printer";
3593 const papi_attribute_t** attrs = NULL;
3594 const papi_job_ticket_t* ticket = NULL;
3595 papi_stream_t stream = NULL;
3596 papi_job_t job = NULL;
3597 char buffer[4096];
3598 size_t buflen = 0;
3599
3600 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3601                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
3602 if (status != PAPI_OK)
3603 {
3604     /* handle the error */
3605     ...
3606 }
3607
3608 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3609                          PAPI_STRING, 1, "test job");
3610 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3611                          PAPI_INTEGER, 1, 4);
3612
3613 /* Open the print job stream */
3614 status = papiJobStreamOpen(handle,
3615                          printer,
3616                          attrs,
3617                          ticket,
3618                          &stream);
3619
3620 if (status != PAPI_OK)
3621 {
3622     fprintf(stderr, "papiJobStreamOpen failed: %s\n",
3623           papiStatusString(status));
3624     ...
3625 }
3626
3627 /* Write all the print job data */
3628 while(print_data_remaining)
3629 {
3630     /* Generate the print data */
3631     ...
3632     /* Write the print data */
3633     status = papiJobStreamWrite(handle
3634                               stream,
3635                               buffer,
3636                               buflen);

```



```

3636         if (status != PAPI_OK)
3637         {
3638             fprintf(stderr, "papiJobStreamWrite failed: %s\n",
3639                     papiStatusString(status));
3640             ...
3641         }
3642     }
3643
3644     /* Close the print job stream */
3645     status = papiJobStreamClose(handle, stream, &job);
3646     if (status != PAPI_OK)
3647     {
3648         fprintf(stderr, "papiJobStreamClose failed: %s\n",
3649                 papiStatusString(status));
3650         ...
3651     }
3652
3653     papiJobFree(job);
3654     papiServiceDestroy(handle);
3655

```

3656

3657

See Also

3658

papiJobStreamWrite, papiJobStreamClose

3659 **7.5. papiJobStreamWrite**

3660

Description

3661

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

3662

3663

Syntax

3664

```

3665     papi_status_t papiJobStreamWrite(
3666                 papi_service_t   handle,
3667                 papi_stream_t     stream,
3668                 const void*       buffer,
3669                 const size_t      buflen );
3670

```

3671

3672

Inputs

3673

3674 handle

Handle to the print service to use.

3675

3676 stream

The open stream object to which print data is written.

3677

3678 buffer

Pointer to the buffer of print data to write.

3679

3680 buflen

The number of bytes to write.

3681

3682

3683

Outputs

3684

none

3685 **Returns**
3686 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3687 value is returned.

3688 **Example**
3689 See papiJobStreamOpen

3690 **See Also**
3691 papiJobStreamOpen, papiJobStreamClose

3692 **7.6. papiJobStreamClose**

3693 **Description**
3694 Closes the specified open job stream and completes submission of the job (if there
3695 were no previous errors returned from papiJobSubmitWrite). The open job stream
3696 must have been obtained by a successful call to papiJobStreamOpen.

3697 **Syntax**

```
3698  
3699            papi_status_t papiJobStreamClose(  
3700                            papi_service_t        handle,  
3701                            papi_stream_t         stream,  
3702                            papi_job_t*           job );  
3703
```

3704

3705 **Inputs**

3706

3707 handle

3708 Handle to the print service to use.

3709 stream

3710 The open stream object to which print data was written.

3711

3712 **Outputs**

3713

3714 job

3715 The resulting job object representing the submitted job.

3716

3717 **Returns**

3718 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3719 value is returned.

3720 **Example**

3721 See papiJobStreamOpen

3722 **See Also**
 3723 papiJobStreamOpen, papiJobStreamWrite

3724 **7.7. papiJobQuery**

3725 **Description**

3726 Queries some or all the attributes of the specified job object.

3727 **Semantics Reference**

3728 Get-Job-Attributes in [RFC2911], section 3.3.4

3729 **Syntax**

3730

```

3731           papi_status_t papiJobQuery(
3732                           papi_service_t     handle,
3733                           const char*        printer_name,
3734                           const int32_t      job_id,
3735                           const char*        requested_attrs[],
3736                           papi_job_t*       job );
3737
```

3738

3739 **Inputs**

3740

3741 handle

3742 Handle to the print service to use.

3743 printer_name

3744 Pointer to the name or URI of the printer to which the job was submitted.

3745 job_id

3746 The ID number of the job to be queried.

3747 requested_attrs

3748 NULL terminated array of attributes to be queried. If NULL is passed then all
 3749 available attributes are queried. (NOTE: The job may return more attributes
 3750 than you requested. This is merely an advisory request that may reduce the
 3751 amount of data returned if the printer/server supports it.)

3752

3753 **Outputs**

3754

3755 job

3756 The returned job object containing the requested attributes.

3757

3758 **Returns**

3759 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3760 value is returned.

3761

Example

3762

```

3763 #include "papi.h"
3764
3765 papi_status_t status;
3766 papi_service_t handle = NULL;
3767 const char* printer_name = "my-printer";
3768 papi_job_t job = NULL;
3769 int32_t job_id = 12;
3770 const char* job_attrs[] =
3771 {
3772     "job-id",
3773     "job-name",
3774     "job-originating-user-name",
3775     "job-state",
3776     "job-state-reasons",
3777     NULL
3778 };
3779 ...
3780 status = papiServiceCreate(&handle,
3781                             NULL,
3782                             NULL,
3783                             NULL,
3784                             NULL,
3785                             PAPI_ENCRYPT_NEVER,
3786                             NULL);
3787
3788 if (status != PAPI_OK)
3789 {
3790     /* handle the error */
3791     ...
3792 }
3793
3794 status = papiJobQuery(handle,
3795                       printer_name,
3796                       job_id,
3797                       job_attrs,
3798                       &job);
3799
3800 if (status != PAPI_OK)
3801 {
3802     /* handle the error */
3803     fprintf(stderr, "papiJobQuery failed: %s\n",
3804            papiServiceGetStatusMessage(handle));
3805     ...
3806 }
3807
3808 if (job != NULL)
3809 {
3810     /* process the job */
3811     ...
3812     papiJobFree(job);
3813 }
3814 papiServiceDestroy(handle);

```

3815

3816

See Also

3817

papiJobFree, papiPrinterListJobs, papiJobModify

3818

7.8. papiJobModify

3819

Description

3820

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

3821

3822

3823

Semantics Reference

3824

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

3825

Syntax

3826

3827

papi_status_t papiJobModify(

```

3828         papi_service_t    handle,
3829         const char*        printer_name,
3830         const int32_t      job_id,
3831         const papi_attribute_t** attrs,
3832         papi_job_t*        job );
3833

```

3834

3835 **Inputs**

3836

3837 handle

3838 Handle to the print service to use.

3839 printer_name

3840 Pointer to the name or URI of the printer to which the job was submitted.

3841 job_id

3842 The ID number of the job to be modified.

3843 attrs

3844 Attributes to be modified. Any attributes not specified are left unchanged.

3845

3846 **Outputs**

3847

3848 job

3849 The modified job object.

3850

3851 **Returns**3852 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3853 value is returned.3854 **Example**

3855

```

3856 #include "papi.h"
3857
3858 papi_status_t status;
3859 papi_service_t handle = NULL;
3860 const char* printer_name = "my-printer";
3861 papi_job_t job = NULL;
3862 int32_t job_id = 12;
3863 papi_attribute_t** attrs = NULL;
3864 ...
3865 status = papiServiceCreate(&handle,
3866                            NULL,
3867                            NULL,
3868                            NULL,
3869                            NULL,
3870                            PAPI_ENCRYPT_NEVER,
3871                            NULL);
3872
3873 if (status != PAPI_OK)
3874 {
3875     /* handle the error */
3876     ...
3877 }
3878 papiAttributeListAddInteger(&attrs,

```

```

3900         PAPI_EXCL,
3901         "copies",
3902         3);
3903
3904     status = papiJobModify(handle,
3905                           printer_name,
3906                           job_id,
3907                           attrs,
3908                           &job);
3909     if (status != PAPI_OK)
3910     {
3911         /* handle the error */
3912         fprintf(stderr, "papiJobModify failed: %s\n",
3913                papiServiceGetStatusMessage(handle));
3914         ...
3915     }
3916     if (job != NULL)
3917     {
3918         /* process the job */
3919         ...
3920         papiJobFree(job);
3921     }
3922     papiServiceDestroy(handle);

```

3905

3906

See Also

3907

papiJobQuery, papiJobFree, papiPrinterListJobs

3908

7.9. papiJobCancel

3909

Description

3910

Cancel the specified print job.

3911

Semantics Reference

3912

Cancel-Job in [RFC2911], section 3.3.3

3913

Syntax

3914

3915

3916

3917

3918

3919

```

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

```

3920

3921

Inputs

3922

3923

handle

3924

Handle to the print service to use.

3925

printer_name

3926

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

3927

job_id

3928

The ID number of the job to be cancelled.

3929

3930

Outputs

3931

none

3932

Returns

3933

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

3934

3935

Example

3936

3937

```

3938 #include "papi.h"
3939
3940 papi_status_t status;
3941 papi_service_t handle = NULL;
3942 const char* printer_name = "my-printer";
3943 int32_t job_id = 12;
3944 ...
3945 status = papiServiceCreate(&handle,
3946                          NULL,
3947                          NULL,
3948                          NULL,
3949                          NULL,
3950                          PAPI_ENCRYPT_NEVER,
3951                          NULL);
3952
3953 if (status != PAPI_OK)
3954 {
3955     /* handle the error */
3956     ...
3957 }
3958
3959 status = papiJobCancel(handle,
3960                      printer_name,
3961                      job_id);
3962
3963 if (status != PAPI_OK)
3964 {
3965     /* handle the error */
3966     fprintf(stderr, "papiJobCancel failed: %s\n",
3967            papiServiceGetStatusMessage(handle));
3968     ...
3969 }
3970
3971 papiServiceDestroy(handle);

```

3970

3971

See Also

3972

papiPrinterListJobs, papiPrinterPurgeJobs

3973

7.10. papiJobHold

3974

Description

3975

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

3976

3977

3978

3979

Semantics Reference

3980

Hold-Job in [RFC2911], section 3.3.5

3981

Syntax

3982

3983

```

3984 papi_status_t papiJobHold(
3985     papi_service_t handle,
3986     const char* printer_name,
3987     const int32_t job_id,
3988     const char* hold_until,

```

```

3988         const time_t*           hold_until_time );
3989

```

3990

3991 **Inputs**

3992

3993 handle

3994 Handle to the print service to use.

3995 printer_name

3996 Pointer to the name or URI of the printer to which the job was submitted.

3997 job_id

3998 The ID number of the job to be held.

3999 hold_until

4000 (optional) Specifies the time when the job will be automatically released for
 4001 printing. If NULL, the job is held until explicitly released by calling
 4002 papiJobRelease. If specified, the value must be one of the strings "indefinite"
 4003 (same effect as passing NULL), "day-time", "evening", "night", "weekend",
 4004 "second-shift", "third-shift", or "timed". For values other than "indefinite" and
 4005 "timed", the printer/server must define exact times associated with these
 4006 values and it may make these associations configurable. If "timed" is specified,
 4007 then the hold_until_time argument is used.

4008 hold_until_time

4009 (optional) Specifies the time when the job will be automatically released for
 4010 printing. This argument is ignored unless "timed" is passed as the hold_until
 4011 argument.

4012

4013 **Outputs**

4014 none

4015 **Returns**

4016 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 4017 value is returned.

4018 **Example**

4019

```

4020 #include "papi.h"
4021
4022 papi_status_t status;
4023 papi_service_t handle = NULL;
4024 const char* printer_name = "my-printer";
4025 int32_t job_id = 12;
4026 ...
4027 status = papiServiceCreate(&handle,
4028                          NULL,
4029                          NULL,
4030                          NULL,
4031                          NULL,
4032                          PAPI_ENCRYPT_NEVER,
4033                          NULL);
4034 if (status != PAPI_OK)
4035 {

```



```

4036         /* handle the error */
4037         ...
4038     }
4039
4040     status = papiJobHold(handle,
4041                        printer_name,
4042                        job_id,
4043                        NULL,
4044                        NULL);
4045     if (status != PAPI_OK)
4046     {
4047         /* handle the error */
4048         fprintf(stderr, "papiJobHold failed: %s\n",
4049                papiServiceGetStatusMessage(handle));
4050         ...
4051     }
4052
4053     papiServiceDestroy(handle);
4054

```

4055

4056 **See Also**4057 `papiJobRelease`4058 **7.11. papiJobRelease**4059 **Description**

4060 Releases the specified print job, allowing it to be scheduled for printing. This
 4061 operation is optional and may not be supported by all printers/servers, but it must
 4062 be supported if `papiJobHold` is supported.

4063 **Semantics Reference**

4064 Release-Job in [RFC2911], section 3.3.6

4065 **Syntax**

4066

```

4067 papi_status_t papiJobRelease(
4068             papi_service_t      handle,
4069             const char*         printer_name,
4070             const int32_t       job_id );
4071

```

4072

4073 **Inputs**

4074

4075 `handle`

4076 Handle to the print service to use.

4077 `printer_name`

4078 Pointer to the name or URI of the printer to which the job was submitted.

4079 `job_id`

4080 The ID number of the job to be released.

4081

4082 **Outputs**4083 `none`

4084

Returns

4085

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

4086

4087

Example

4088

4089

```

4089 #include "papi.h"
4090
4091 papi_status_t status;
4092 papi_service_t handle = NULL;
4093 const char* printer_name = "my-printer";
4094 int32_t job_id = 12;
4095 ...
4096 status = papiServiceCreate(&handle,
4097                             NULL,
4098                             NULL,
4099                             NULL,
4100                             NULL,
4101                             PAPI_ENCRYPT_NEVER,
4102                             NULL);
4103
4104 if (status != PAPI_OK)
4105 {
4106     /* handle the error */
4107     ...
4108 }
4109
4110 status = papiJobRelease(handle,
4111                         printer_name,
4112                         job_id);
4113
4114 if (status != PAPI_OK)
4115 {
4116     /* handle the error */
4117     fprintf(stderr, "papiJobRelease failed: %s\n",
4118             papiServiceGetStatusMessage(handle));
4119     ...
4120 }
4121 papiServiceDestroy(handle);

```

4122

4123

See Also

4124

papiJobHold

4125

7.12. papiJobRestart

4126

Description

4127

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

4128

4129

4130

Semantics Reference

4131

Restart-Job in [RFC2911], section 3.3.7

4132

Syntax

4133

4134

```

4134 papi_status_t papiJobRestart(
4135             papi_service_t handle,
4136             const char* printer_name,
4137             const int32_t job_id );
4138

```

4138

4139

4140 **Inputs**

4141

4142 handle

4143 Handle to the print service to use.

4144 printer_name

4145 Pointer to the name or URI of the printer to which the job was submitted.

4146 job_id

4147 The ID number of the job to be restarted.

4148

4149 **Outputs**

4150 none

4151 **Returns**4152 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
4153 value is returned.4154 **Example**

4155

```

4156 #include "papi.h"
4157
4158 papi_status_t status;
4159 papi_service_t handle = NULL;
4160 const char* printer_name = "my-printer";
4161 int32_t job_id = 12;
4162 ...
4163 status = papiServiceCreate(&handle,
4164                             NULL,
4165                             NULL,
4166                             NULL,
4167                             NULL,
4168                             PAPI_ENCRYPT_NEVER,
4169                             NULL);
4170
4171 if (status != PAPI_OK)
4172 {
4173     /* handle the error */
4174     ...
4175 }
4176
4177 status = papiJobRestart(handle,
4178                         printer_name,
4179                         job_id);
4180
4181 if (status != PAPI_OK)
4182 {
4183     /* handle the error */
4184     fprintf(stderr, "papiJobRestart failed: %s\n",
4185             papiServiceGetStatusMessage(handle));
4186     ...
4187 }
4188 papiServiceDestroy(handle);

```

4189

4190 **See Also**

4191 papiPrinterListJobs

4192 **7.13. papiJobGetAttributeList**4193 **Description**

4194 Get the attribute list associated with a job object.

4195 This function retrieves an attribute list from a job object returned in a previous call.
 4196 Job objects are returned as a result of the operations performed by
 4197 `papiPrinterListJobs`, `papiJobQuery`, `papiJobModify`, `papiJobSubmit`,
 4198 `papiJobSubmitByReference`, and `papiJobClose`.

4199 **Syntax**

4200

```
4201 papi_attribute_t** papiJobGetAttributeList(  
4202     papi_job_t     job );  
4203
```

4204

4205 **Inputs**

4206

4207 job

Handle of the job object.

4209

4210 **Outputs**

4211 none

4212 **Returns**

4213 Pointer to the attribute list associated with the job object.

4214 **Example**

4215

```
4216 #include "papi.h"  
4217  
4218 papi_status_t status;  
4219 papi_service_t handle = NULL;  
4220 const char* printer_name = "my-printer";  
4221 papi_job_t job = NULL;  
4222 papi_attribute_list* attrs = NULL;  
4223 ...  
4224 status = papiServiceCreate(&handle,  
4225     NULL,  
4226     NULL,  
4227     NULL,  
4228     NULL,  
4229     PAPI_ENCRYPT_NEVER,  
4230     NULL);  
4231  
4232 if (status != PAPI_OK)  
4233 {  
4234     /* handle the error */  
4235     ...  
4236 }  
4237  
4238 status = papiJobQuery(handle,  
4239     printer_name,  
4240     67,  
4241     NULL,  
4242     &job);  
4243  
4244 if (status != PAPI_OK)  
4245 {  
4246     /* handle the error */  
4247     fprintf(stderr, "papiJobQuery failed: %s\n",  
4248         papiServiceGetStatusMessage(handle));  
4249     ...  
4250 }  
4251  
4252 if (job != NULL)  
4253 {  
4254     /* process the job object */  
4255     attrs = papiJobGetAttributeList(job);  
4256     ...  
4257     papiJobFree(job);  
4258 }
```

```

4256     }
4257
4258     papiServiceDestroy(handle);
4259

```

4260

4261 **See Also**

4262 papiPrinterListJobs, papiJobQuery

4263 **7.14. papiJobGetPrinterName**4264 **Description**

4265 Get the printer name associated with a job object.

4266 **Syntax**

4267

```

4268 char* papiJobGetPrinterName(
4269     papi_job_t job );
4270

```

4271

4272 **Inputs**

4273

4274 job

4275 Handle of the job object.

4276

4277 **Outputs**

4278 none

4279 **Returns**

4280 Pointer to the printer name associated with the job object.

4281 **Example**

4282

```

4283 #include "papi.h"
4284
4285 char* printer_name = NULL;
4286 papi_job_t job = NULL;
4287 ...
4288 if (job != NULL)
4289 {
4290     /* process the job object */
4291     printer_name = papiJobGetPrinterName(job);
4292     ...
4293     papiJobFree(job);
4294 }
4295

```

4296

4297 **See Also**

4298 papiPrinterListJobs, papiJobQuery

4299 **7.15. papiJobGetId**

4300 **Description**

4301 Get the job ID associated with a job object.

4302 **Syntax**

4303

```
4304 int32_t papiJobGetId(  
4305         papi_job_t    job );  
4306
```

4307

4308 **Inputs**

4309

4310 job

Handle of the job object.

4312

4313 **Outputs**

4314 none

4315 **Returns**

4316 The job ID associated with the job object.

4317 **Example**

4318

```
4319 #include "papi.h"  
4320  
4321 int32_t job_id;  
4322 papi_job_t job = NULL;  
4323 ...  
4324 if (job != NULL)  
4325 {  
4326     /* process the job object */  
4327     job_id = papiJobGetId(job);  
4328     ...  
4329     papiJobFree(job);  
4330 }  
4331
```

4332

4333 **See Also**

4334 papiPrinterListJobs, papiJobQuery

4335 **7.16. papiJobGetJobTicket**

4336 **Description**

4337 Get the job ticket associated with a job object.

4338 **Syntax**

4339

```
4340 papi_job_ticket_t* papiJobGetJobTicket(  
4341         papi_job_t    job );  
4342
```

4343

4344 **Inputs**

4345

4346 job

4347 Handle of the job object.

4348

4349 **Outputs**

4350 none

4351 **Returns**

4352 Pointer to the job ticket associated with the job object.

4353 **Example**

4354

```

4355           #include "papi.h"
4356
4357           papi_job_ticket_t* job_ticket = NULL;
4358           papi_job_t job = NULL;
4359           ...
4360           if (job != NULL)
4361           {
4362               /* process the job object */
4363               job_ticket = papiJobGetJobTicket(job);
4364               ...
4365               papiJobFree(job);
4366           }
4367

```

4368

4369 **See Also**

4370 papiPrinterListJobs, papiJobQuery

4371 **7.17. papiJobFree**4372 **Description**

4373 Free a job object.

4374 **Syntax**

4375

```

4376           void papiJobFree(
4377                            papi_job_t        job );
4378

```

4379

4380 **Inputs**

4381

4382 job

4383 Handle of the job object to free.

4384

4385

Outputs

4386

none

4387

Returns

4388

none

4389

Example

4390

```

4391 #include "papi.h"
4392
4393 papi_status_t status;
4394 papi_service_t handle = NULL;
4395 const char* printer_name = "my-printer";
4396 papi_job_t job = NULL;
4397 ...
4398 status = papiServiceCreate(&handle,
4399                            NULL,
4400                            NULL,
4401                            NULL,
4402                            NULL,
4403                            PAPI_ENCRYPT_NEVER,
4404                            NULL);
4405
4406 if (status != PAPI_OK)
4407 {
4408     /* handle the error */
4409     ...
4410 }
4411
4412 status = papiJobQuery(handle,
4413                      printer_name,
4414                      12,
4415                      &job);
4416
4417 if (status != PAPI_OK)
4418 {
4419     /* handle the error */
4420     fprintf(stderr, "papiJobQuery failed: %s\n",
4421            papiServiceGetStatusMessage(handle));
4422     ...
4423 }
4424
4425 if (job != NULL)
4426 {
4427     /* process the job object */
4428     ...
4429     papiJobFree(job);
4430 }
4431
4432 papiServiceDestroy(handle);

```

4432

4433

See Also

4434

papiJobQuery

4435 **7.18. papiJobListFree**

4436

Description

4437

Free a list of job objects.

4438

Syntax

4439

4440

```

void papiJobListFree(
4441     papi_job_t* jobs );
4442

```

4443

4443

4444

Inputs

4445

4446 jobs

Pointer to the job object list to free.

4448

4449

Outputs

4450

none

4451

Returns

4452

none

4453

Example

4454

```

4455 #include "papi.h"
4456
4457 papi_status_t status;
4458 papi_service_t handle = NULL;
4459 const char* printer_name = "my-printer";
4460 papi_job_t* jobs = NULL;
4461 ...
4462 status = papiServiceCreate(&handle,
4463                          NULL,
4464                          NULL,
4465                          NULL,
4466                          NULL,
4467                          PAPI_ENCRYPT_NEVER,
4468                          NULL);
4469
4470 if (status != PAPI_OK)
4471 {
4472     /* handle the error */
4473     ...
4474 }
4475
4476 status = papiPrinterListJobs(handle,
4477                             printer_name,
4478                             NULL,
4479                             0, 0, 0,
4480                             &jobs);
4481
4482 if (status != PAPI_OK)
4483 {
4484     /* handle the error */
4485     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
4486            papiServiceGetStatusMessage(handle));
4487     ...
4488 }
4489
4490 if (jobs != NULL)
4491 {
4492     /* process the job objects */
4493     ...
4494     papiJobListFree(jobs);
4495 }
4496
4497 papiServiceDestroy(handle);

```

4497

4498

See Also

4499

papiPrinterListJobs

4500 Chapter 8. Miscellaneous API

4501 8.1. papiStatusString

4502 Description

4503 Get a status string for the specified papi_status_t. The status message returned
4504 from this function may be less detailed than the status message returned from
4505 papiServiceGetStatusMessage (if the print service supports returning more detailed
4506 error messages).

4507 The returned message will be localized in the language of the submitter of the
4508 requestor.

4509 Syntax

4510

```
4511 char* papiStatusString(  
4512     const papi_status_t status );  
4513
```

4514

4515 Inputs

4516

4517 status

4518 The status value to convert to a status string.

4519

4520 Outputs

4521 none

4522 Returns

4523 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
4524 value is returned.

4525 Example

4526

```
4527 #include "papi.h"  
4528 papi_status_t status;  
4529 ...  
4530 fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));  
4531  
4532
```

4533

4534 See Also

4535 papiServiceGetStatusMessage

4536 Chapter 9. Attributes

4537 For a summary of the IPP attributes which can be used with the PAPI interface, see:
4538 <ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf>

4539 9.1. Extension Attributes

4540 The following attributes are not currently defined by IPP, but may be used with
4541 this API.

4542 9.1.1. job-ticket-formats-supported

4543 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats
4544 that are supported by the printer. If this attribute is not present, it is assumed that
4545 the printer does not support any job ticket formats.

4546

4547 9.1.2. media-margins

4548 (1setOf integer) The media-margins attribute defines the printable margins for the
4549 current printer object and consists of exactly 4 or 8 ordered integers. Each group of 4
4550 integers represent the minimum distance from the left, bottom, right, and top edges
4551 of the media in micrometers.

4552 If 4 integers are provided, the margins are the same for the front and back sides of
4553 the media when producing duplexed output. If 8 integers are provided, the first 4
4554 integers represent the margins for the front side and the last 4 integers represent the
4555 margins for the back side of the media.

4556 Unless otherwise specified, the margin values represent the minimum margins that
4557 can be used with all sizes and types of media.

4558 9.2. Required Job Attributes

4559 The following job attributes *must* be supported to comply with this API standard.
4560 These attributes may be supported by the underlying print server directly, or they
4561 may be mapped by the PAPI library.

job-id
job-name
job-originating-user-name
job-printer-uri
job-state
job-state-reasons
job-uri
time-at-creation
time-at-processing
time-at-completed

4562

4563 9.3. Required Printer Attributes

4564 The following printer attributes *must* be supported to comply with this API
4565 standard. These attributes may be supported by the underlying print server
4566 directly, or they may be mapped by the PAPI library.

charset-configured
charset-supported

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

4567

4568 **9.4. IPP Attribute Type Mapping**

4569

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
mediaType	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

4570

4571 Appendix A. Attribute List Text Representation

4572 A.1. ABNF Definition

4573 The following ABNF definition [RFC2234] describes the syntax of PAPI attributes
4574 encoded as text options:

```
4575 OPTION-STRING = [OPTION] *(1*WC OPTION) *WC
4576
4577 OPTION          = TRUEOPTION / FALSEOPTION / VALUEOPTION
4578
4579 TRUEOPTION     = NAME
4580
4581 FALSEOPTION    = "no" NAME
4582
4583 VALUEOPTION    = NAME "=" VALUE *( "," VALUE )
4584
4585 NAME           = 1*NAMECHAR
4586
4587 NAMECHAR       = DIGIT / ALPHA / "-" / "_" / "."
4588
4589 VALUE          = BOOLVALUE / COLVALUE / DATEVALUE / NUMBEVALUE / QUOTEDVALUE /
4590                 RANGEVALUE / RESVALUE / STRINGVALUE
4591
4592 BOOLVALUE      = "yes" / "no" / "true" / "false"
4593
4594 COLVALUE       = "{" OPTION-STRING "}"
4595
4596 DATEVALUE      = HOUR MINUTE [ SECOND ] / YEAR MONTH DAY /
4597                 YEAR MONTH DAY HOUR MINUTE [ SECOND ]
4598
4599 YEAR           = 4DIGIT
4600
4601 MONTH          = "0" %x31-39 / "10" / "11" / "12"
4602
4603 DAY            = %x30-32 DIGIT / "1" DIGIT / "2" DIGIT / "30" / "31"
4604
4605 HOUR           = %x30-31 DIGIT / "1" DIGIT / "20" / "21" / "22" / "23"
4606
4607 MINUTE         = %x30-35 DIGIT
4608
4609 SECOND         = %x30-35 DIGIT
4610
4611 NUMBEVALUE     = 1*DIGIT / "-" 1*DIGIT / "+" 1*DIGIT
4612
4613 QUOTEDVALUE    = DQUOTE *QUOTEDCHAR DQUOTE / SQUOTE *QUOTEDCHAR SQUOTE
4614
4615 QUOTEDCHAR     = %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
4616                 %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
4617                 %x5D-7E / %xA0-FF
4618
4619 OCTALDIGIT     = %x30-37
4620
4621 RANGEVALUE     = 1*DIGIT "-" 1*DIGIT
4622
4623 RESVALUE       = 1*DIGIT [ "x" 1*DIGIT ] ("dpi" / "dpc")
4624
4625 STRINGVALUE    = 1*STRINGCHAR
4626
4627 STRINGCHAR     = %x5C %x20 / %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
4628                 %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
4629                 %x5D-7E / %xA0-FF
4630
4631 SQUOTE         = %x27
4632
4633 WC             = %x09 / %x0A / %x20
4634
```

4635 A.2. Examples

4636 The following example strings illustrate the format of text options:

4637 Boolean Attributes:

```
4638 foo
4639 nofoo
4640 foo=false
4641 foo=true
4642 foo=no
4643 foo=yes
4644
```

Appendix A. Attribute List Text Representation

4645 **Collection Attributes:**
~~4646~~
~~4647~~ media-col={media-size={x-dimension=123 y-dimension=456}}

4648 **Integer Attributes:**
~~4649~~
~~4650~~ copies=123
~~4651~~ hue=-123

4652 **String Attributes:**
~~4653~~
~~4654~~ job-sheets=standard
~~4655~~ job-sheets=standard, standard
~~4656~~ media=na-custom-foo.8000-10000
~~4657~~ job-name=John\'s\ Really\040Nice\ Document

4658 **String Attributes (quoted):**
~~4659~~
~~4660~~ job-name="John\'s Really Nice Document"
~~4661~~ document-name='Another \'Word\042 document.doc'

4662 **Range Attributes:**
~~4663~~
~~4664~~ page-ranges=1-5
~~4665~~ page-ranges=1-2,5-6,101-120

4666 **Date Attributes:**
~~4667~~
~~4668~~ job-hold-until-datetime=1234
~~4669~~ job-hold-until-datetime=123456
~~4670~~ job-hold-until-datetime=20020904
~~4671~~ job-hold-until-datetime=200209041234
~~4672~~ job-hold-until-datetime=20020904123456

4673 **Resolution Attributes:**
~~4674~~
~~4675~~ resolution=360dpi
~~4676~~ resolution=720x360dpi
~~4677~~ resolution=1000dpc

4678 **Multiple Attributes:**
~~4679~~
~~4680~~ job-sheets=standard page-ranges=1-2,5-6,101-120 resolution=360dpi

4681 **Appendix B. References**

4682 **B.1. Internet Printing Protocol (IPP)**

4683 IETF RFCs can be obtained from "<http://www.rfc-editor.org/rfcsearch.html>".
4684 Other IPP documents can be obtained from
4685 "<http://www.pwg.org/ipp/index.html>" and
4686 "ftp://ftp.pwg.org/pub/pwg/ipp/new_XXX/".

4687 [RFC2911] T. Hastings, R. Herriot, R. deBry, S. Isaacson, and P. Powell August 1998
4688 *Internet Printing Protocol/1.1: Model and Semantics* (Obsoletes 2566)

4689 [RFC3196] T. Hastings, H. Holst, C. Kugler, C. Manros, and P. Zehler November
4690 2001 *Internet Printing Protocol/1.1: Implementor's Guide*

4691 [RFC3380] T. Hastings, R. Herriot, C. Kugler, and H. Lewis September 2002 *Internet*
4692 *Printing Protocol (IPP): Job and Printer Set Operations*

4693 [RFC3381] T. Hastings, H. Lewis, and R. Bergman September 2002 *Internet Printing*
4694 *Protocol (IPP): Job Progress Attributes*

4695 [RFC3382] R. deBry, T. Hastings, R. Herriot, K. Ocke, and P. Zehler September 2002
4696 *Internet Printing Protocol (IPP): The 'collection' attribute syntax*

4697 [5100.2] T. Hastings and R. Bergman IEEE-ISTO 5100.2 February 2001 *Internet*
4698 *Printing Protocol (IPP): output-bin attribute extension*

4699 [5100.3] T. Hastings and K. Ocke IEEE-ISTO 5100.3 February 2001 *Internet Printing*
4700 *Protocol (IPP): Production Printing Attributes*

4701 [5100.4] R. Herriot and K. Ocke IEEE-ISTO 5100.4 February 2001 *Internet Printing*
4702 *Protocol (IPP): Override Attributes for Documents and Pages*

4703 [5101.1] T. Hastings and D. Fullman IEEE-ISTO 5101.1 February 2001 *Internet*
4704 *Printing Protocol (IPP): finishings attribute values extension*

4705 [ops-set2] C. Kugler, T. Hastings, and H. Lewis July 2001 *Internet Printing Protocol*
4706 *(IPP): Job and Printer Administrative Operations*

4707 **B.2. Job Ticket**

4708 [jdf] CIP4 Organization April 2002 *Job Definition Format (JDF) Specification Version*
4709 *1.1*

4710 **B.3. Printer Working Group (PWG)**

4711 [PWGSemMod] P. Zehler and Albright September 2002 *Printer Working Group*
4712 *(PWG): Semantic Model*

4713 **B.4. Other**

4714 [RFC1738] T. Berners-Lee, L. Masinter, and M. McCahill December 1994 *Uniform*
4715 *Resource Locators (URL)* (Updated by RFC1808, RFC2368, RFC2396)

4716 [RFC2234] D. Crocker and P. Overell November 1997 *Augmented BNF for Syntax*
4717 *Specifications: ABNF*

4718 [RFC2396] T. Berners-Lee, R. Fielding, and L. Masinter August 1998 *Uniform*
4719 *Resource Locators (URL): Generic Syntax* (Updates RFC1808, RFC1738)

4720 **Appendix C. Change History**

4721 **Version 0.8 (November 15, 2002)**

4722

- 4723 • Added value field, explanation, and corrected example for `papi_filter_t`.
- 4724 • Added `media-margins` attribute to "Extension Attributes" section.
- 4725 • Renamed function names with "Username" to "UserName", and renamed
- 4726 function names with "Servicename" to "ServiceName", and
- 4727 • Miscellaneous wording and typo corrections.

4728

4729 **Version 0.7 (October 18, 2002)**

4730

- 4731 • Added `attr_delim` argument to `papiAttributeListToString` and made new-line
- 4732 (`"\n"`) an allowed attribute delimiter on input to `papiAttributeListFromString`.
- 4733 • Added "Semantics Reference" subsections to functions.
- 4734 • Added to References: [5101.1], [RFC3196], and URIs for obtaining IPP
- 4735 documents.
- 4736 • Added `PAPI_JOB_TICKET_NOT_SUPPORTED` status code.
- 4737 • Added "Globalization" section in the "Print System Model" chapter.
- 4738 • Changed definition and usage of returned value from
- 4739 `papiAttributeListGetValue`. Also clarified what happens to output values when a
- 4740 `papiAttributeListGet*` call has an error.
- 4741 • Clarified descriptions of `papiPrinterGetAttributeList` and
- 4742 `papiJobGetAttributeList`.
- 4743 • Changed buffer length arguments from `int` to `size_t`.
- 4744 • Clarified that `papiServiceDestroy` must always be called after a call to
- 4745 `papiServiceCreate`.
- 4746 • Removed `attributes-charset`, `attributes-natural-language`, and `job-printer-up-time`
- 4747 from the "Required Job Attributes" (they may be hidden inside the PAPI
- 4748 implementation).
- 4749 • Clarified result of passing both attributes and a job ticket on all the job
- 4750 submission functions.
- 4751 • Miscellaneous wording and typo corrections.

4752

4753 **Version 0.6 (September 20, 2002)**

4754

- 4755 • Made explanation of `requested_attrs` in `papiPrintersList` the same as it is for
- 4756 `papiPrinterQuery`.
- 4757 • Moved `units` argument on `papiAttributeListAddResolution` to the end of the
- 4758 argument list to match the corresponding get function.
- 4759 • Added `papiAttributeListAddCollection` and `papiAttributeListGetCollection`.

- 4760 • Removed unneeded extra level of indirection from attrs argument to
- 4761 papiAttributeListGet* functions. Also made the attrs argument const.
- 4762 • Added note to "Conventions" section that strings are assumed to be UTF-8
- 4763 encoded.
- 4764 • Added papiAttributeListFromString and papiAttributeListToString functions,
- 4765 along with a new appendix defining the string format syntax.
- 4766 • Added papiJobSubmitByReference, papiJobStreamOpen, papiJobStreamWrite,
- 4767 and papiJobStreamClose functions.
- 4768 • Added short "Document" section in the "Print System Model" chapter.
- 4769 • Added explanation of how multiple files specified in the papiJobSubmit
- 4770 file_names argument are handled by the print system.
- 4771 • Changed papi_job_ticket_t "uri" field to "file_name" and added explanation text.
- 4772 • Added explanation of implementation option for merging papiJobSubmit
- 4773 attributes with job_ticket argument.
- 4774 • Added "References" appendix.
- 4775 • Added "IPP Attribute Type Mapping" appendix.
- 4776 • Added "PWG" job ticket format to papi_jt_format_t.
- 4777 • Miscellaneous wording and typo corrections.
- 4778

Version 0.5 (August 30, 2002)

- 4779
- 4780
- 4781 • Added job_attrs argument to papiPrinterQuery to support more accurate query
- 4782 of printer capabilities.
- 4783 • Added management functions papiAttributeDelete, papiJobModify, and
- 4784 papiPrinterModify.
- 4785 • Added functions papiAttributeListGetValue, papiAttributeListGetString,
- 4786 papiAttributeListGetInteger, etc.
- 4787 • Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent
- 4788 with the naming convention (first word after "papi" is the object being operated
- 4789 upon).
- 4790 • Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.
- 4791 • Made description of authentication more implementation-independent.
- 4792 • Added reference to IPP attributes summary document.
- 4793 • Added result argument to papiPrinterPurgeJobs.
- 4794 • Added "collection attribute" support (PAPI_COLLECTION type).
- 4795 • Changed boolean values to consistently use char. Added PAPI_FALSE and
- 4796 PAPI_TRUE enum values.
- 4797

Version 0.4 (July 19, 2002)

4798
4799

- 4800 • Made papi_job_t and papi_printer_t opaque handles and added "get" functions
- 4801 to access the associated information (papiPrinterGetAttributeList,
- 4802 papiJobGetAttributeList, papiJobGetId, papiJobGetPrinterName,
- 4803 papiJobGetJobTicket).
- 4804 • Removed variable length argument lists from attribute add functions.
- 4805 • Changed order and name of flag value passed to attribute add functions.
- 4806 • Eliminated indirection in date/time value passed to papiAttributeAddDatetime.
- 4807 • Added message argument to papiPrinterPause.
- 4808

Version 0.3 (June 24, 2002)

- 4809
- 4810
- 4811 • Converted to DocBook format from Microsoft Word
- 4812 • Major rewrite, including:
 - 4813 • Changed how printer names are described in "Model/Printer"
 - 4814 • Changed fixed length strings to pointers in numerous structures/sections
 - 4815 • Redefined attribute/value structures and associated API descriptions
 - 4816 • Changed list/query functions to return "objects"
 - 4817 • Rewrote "Attributes API" chapter
 - 4818 • Changed many function definitions to pass NULL-terminated arrays of
 - 4819 pointers instead of a separate count argument
 - 4820 • Changed papiJobSubmit to take an attribute list structure as input instead of a
 - 4821 formatted string
- 4822
- 4823

Version 0.2 (April 17, 2002)

- 4824
- 4825
- 4826 • Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)
- 4827 • Filled in "Encryption" section and added information about encryption in "Object
- 4828 Identification" section
- 4829 • Added "short_name" field in "Object Identification" section
- 4830 • Added "Job Ticket (papi_job_ticket_t)" section
- 4831 • Added papiPrinterPause
- 4832 • Added papiPrinterResume
- 4833 • Added papiPurgeJobs
- 4834 • Added optional job_ticket argument to papiJobSubmit
- 4835 • Added optional passing of filenames by URI to papiJobSubmit
- 4836 • Added papiHoldJob
- 4837 • Added papiReleaseJob
- 4838 • Added papiRestartJob

4839

4840

Version 0.1 (April 3, 2002)

4841

4842

- Original draft version

4843

4844

4845

4846

4847

4848

End of Document