

1

2

Open Standard Print API (PAPI)

3

Version 0.7 (DRAFT)

4

5

Alan Hlava

6

IBM Printing Systems Division

7

Norm Jacobs

8

Sun Microsystems, Inc.

9

Michael R Sweet

10

Easy Software Products

11

11

12 **Open Standard Print API (PAPI): Version 0.7 (DRAFT)**

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

14 Version 0.7 (DRAFT) Edition

15 Copyright © 2002 by Free Standards Group

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

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

Table of Contents

1. Introduction.....	1
2. Print System Model	2
2.1. Introduction.....	2
2.2. Model.....	2
2.2.1. Print Service	2
2.2.2. Printer	2
2.2.3. Job.....	3
2.2.4. Document	3
2.3. Security.....	3
2.3.1. Authentication	3
2.3.2. Authorization.....	3
2.3.3. Encryption.....	3
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).....	6
3.8. Status (papi_status_t).....	7
3.9. List Filter (papi_filter_t).....	8
4. Service API	9
4.1. papiServiceCreate	9
4.2. papiServiceDestroy.....	11
4.3. papiServiceSetUsername	12
4.4. papiServiceSetPassword	13
4.5. papiServiceSetEncryption.....	14
4.6. papiServiceSetAuthCB.....	15
4.7. papiServiceSetAppData	16
4.8. papiServiceGetServicename.....	17
4.9. papiServiceGetUsername	18
4.10. papiServiceGetPassword	19
4.11. papiServiceGetEncryption.....	20
4.12. papiServiceGetAppData	21
4.13. papiServiceGetStatusMessage	21
5. Printer API.....	23
5.1. Usage	23
5.2. papiPrintersList.....	23
5.3. papiPrinterQuery.....	25
5.4. papiPrinterModify	27
5.5. papiPrinterPause.....	28
5.6. papiPrinterResume	30
5.7. papiPrinterPurgeJobs	31
5.8. papiPrinterListJobs	32
5.9. papiPrinterGetAttributeList.....	34
5.10. papiPrinterFree	36

67	5.11. papiPrinterListFree.....	37
68	6. Attributes API.....	39
69	6.1. papiAttributeListAdd	39
70	6.2. papiAttributeListAddString.....	40
71	6.3. papiAttributeListAddInteger.....	41
72	6.4. papiAttributeListAddBoolean	42
73	6.5. papiAttributeListAddRange	44
74	6.6. papiAttributeListAddResolution.....	45
75	6.7. papiAttributeListAddDatetime	46
76	6.8. papiAttributeListAddCollection.....	48
77	6.9. papiAttributeDelete.....	49
78	6.10. papiAttributeListGetValue.....	50
79	6.11. papiAttributeListGetString.....	51
80	6.12. papiAttributeListGetInteger.....	53
81	6.13. papiAttributeListGetBoolean.....	54
82	6.14. papiAttributeListGetRange	55
83	6.15. papiAttributeListGetResolution	56
84	6.16. papiAttributeListGetDatetime	58
85	6.17. papiAttributeListGetCollection	59
86	6.18. papiAttributeListFree.....	60
87	6.19. papiAttributeListFind	61
88	6.20. papiAttributeListGetNext.....	62
89	6.21. papiAttributeListFromString	63
90	6.22. papiAttributeListToString	64
91	7. Job API	66
92	7.1. papiJobSubmit.....	66
93	7.2. papiJobSubmitByReference.....	68
94	7.3. papiJobValidate.....	70
95	7.4. papiJobStreamOpen	72
96	7.5. papiJobStreamWrite	74
97	7.6. papiJobStreamClose	75
98	7.7. papiJobQuery	76
99	7.8. papiJobModify	77
100	7.9. papiJobCancel	79
101	7.10. papiJobHold	80
102	7.11. papiJobRelease	82
103	7.12. papiJobRestart	83
104	7.13. papiJobGetAttributeList	84
105	7.14. papiJobGetPrinterName	86
106	7.15. papiJobGetId	87
107	7.16. papiJobGetJobTicket.....	87
108	7.17. papiJobFree.....	88
109	7.18. papiJobListFree	89
110	8. Miscellaneous API	91
111	8.1. papiStatusString.....	91
112	9. Attributes.....	92
113	9.1. Extension Attributes.....	92
114	9.1.1. job-ticket-formats-supported.....	92
115	9.2. Required Job Attributes	92
116	9.3. Required Printer Attributes.....	92
117	9.4. IPP Attribute Type Mapping.....	93

118	A. Attribute List Text Representation	94
119	A.1. ABNF Definition	94
120	A.2. Examples	94
121	B. References	96
122	B.1. Internet Printing Protocol (IPP)	96
123	96
124	B.2. Job Ticket	96
125	96
126	B.3. Printer Working Group (PWG)	96
127	96
128	B.4. Other	96
129	96
130	C. Change History	97

131 **Chapter 1. Introduction**

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

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

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

141

142

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

145 **Chapter 2. Print System Model**

146 **2.1. Introduction**

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

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

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

157

158 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	

159

160

161 **2.2. Model**

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

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

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

173 **2.2.1. Print Service**

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

178 **2.2.2. Printer**

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

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

187 **2.2.3. Job**

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

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

194 **2.2.4. Document**

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

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

202 **2.3. Security**

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

205 **2.3.1. Authentication**

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

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

212 **2.3.2. Authorization**

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

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

221 **2.3.3. Encryption**

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

227 **2.4. Globalization**

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

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

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

238 Chapter 3. Common Structures

239 3.1. Conventions

240

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

246

247 3.2. Service Object (papi_service_t)

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

```
251 typedef void* papi_service_t;
```

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

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

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

265 3.3. Attributes and Values

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

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

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

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

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

```

struct papi_attribute_str;

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

```

```

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

```

```

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

```

338

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

339

3.4. Job Object (`papi_job_t`)

340

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

341

342

343

344

3.5. Stream Object (`papi_stream_t`)

345

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

346

347

3.6. Printer Object (`papi_printer_t`)

348

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

349

350

351

352

3.7. Job Ticket (`papi_job_ticket_t`)

353

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

354

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

```

357 /* Job Ticket Format */
358 typedef enum
359 {
360     PAPI_JT_FORMAT_JDF = 0,      /* Job Definition Format */
361     PAPI_JT_FORMAT_PWG = 1,     /* PWG Job Ticket Format */
362 } papi_jt_format_t;

364 /* Job Ticket */
365 typedef struct papi_job_ticket_s
366 {
367     papi_jt_format_t format;     /* Format of job ticket */
368     char* ticket_data;         /* Buffer containing the job
369                                ticket data. If NULL,
370                                file_name must be specified */
371     char* file_name;           /* Name of the file containing
372                                the job ticket data. If
373                                ticket_data is specified, then
374                                file_name is ignored. */
375 } papi_job_ticket_t;
376

```

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

384 3.8. Status (papi_status_t)

```

385 typedef enum
386 {
387     PAPI_OK = 0x0000,
388     PAPI_OK_SUBST,
389     PAPI_OK_CONFLICT,
390     PAPI_OK_IGNORED_SUBSCRIPTIONS,
391     PAPI_OK_IGNORED_NOTIFICATIONS,
392     PAPI_OK_TOO_MANY_EVENTS,
393     PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
394     PAPI_REDIRECTION_OTHER_SITE = 0x300,
395     PAPI_BAD_REQUEST = 0x0400,
396     PAPI_FORBIDDEN,
397     PAPI_NOT_AUTHENTICATED,
398     PAPI_NOT_AUTHORIZED,
399     PAPI_NOT_POSSIBLE,
400     PAPI_TIMEOUT,
401     PAPI_NOT_FOUND,
402     PAPI_GONE,
403     PAPI_REQUEST_ENTITY,
404     PAPI_REQUEST_VALUE,
405     PAPI_DOCUMENT_FORMAT,
406     PAPI_ATTRIBUTES,
407     PAPI_URI_SCHEME,
408     PAPI_CHARSET,
409     PAPI_CONFLICT,
410     PAPI_COMPRESSION_NOT_SUPPORTED,
411     PAPI_COMPRESSION_ERROR,
412     PAPI_DOCUMENT_FORMAT_ERROR,
413     PAPI_DOCUMENT_ACCESS_ERROR,
414     PAPI_ATTRIBUTES_NOT_SETTABLE,
415     PAPI_IGNORED_ALL_SUBSCRIPTIONS,
416     PAPI_TOO_MANY_SUBSCRIPTIONS,
417     PAPI_IGNORED_ALL_NOTIFICATIONS,
418     PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
419     PAPI_INTERNAL_ERROR = 0x0500,
420     PAPI_OPERATION_NOT_SUPPORTED,
421     PAPI_SERVICE_UNAVAILABLE,
422     PAPI_VERSION_NOT_SUPPORTED,
423     PAPI_DEVICE_ERROR,
424     PAPI_TEMPORARY_ERROR,
425     PAPI_NOT_ACCEPTING,
426     PAPI_PRINTER_BUSY,
427     PAPI_ERROR_JOB_CANCELLED,
428     PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
429     PAPI_PRINTER_IS_DEACTIVATED,

```

```

430     PAPI_BAD_ARGUMENT,
431     PAPI_JOB_TICKET_NOT_SUPPORTED
432 } papi_status_t;
433

```

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

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

439 3.9. List Filter (papi_filter_t)

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

```

443 typedef enum
444 {
445     PAPI_FILTER_BITMASK = 0
446     /* future filter types may be added here */
447 } papi_filter_type_t;
448
449 typedef struct
450 {
451     papi_filter_type_t type; /* Type of filter specified */
452
453     union
454     {
455         unsigned int mask; /* PAPI_FILTER_BITMASK */
456
457         /* future filter types may be added here */
458     } u;
459 } papi_filter_t;
460

```

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

```

463 enum
464 {
465     PAPI_PRINTER_LOCAL = 0x0000, /* Local printer or class */
466     PAPI_PRINTER_CLASS = 0x0001, /* Printer class */
467     PAPI_PRINTER_REMOTE = 0x0002, /* Remote printer or class */
468     PAPI_PRINTER_BW = 0x0004, /* Can do B&W printing */
469     PAPI_PRINTER_COLOR = 0x0008, /* Can do color printing */
470     PAPI_PRINTER_DUPLEX = 0x0010, /* Can do duplexing */
471     PAPI_PRINTER_STAPLE = 0x0020, /* Can staple output */
472     PAPI_PRINTER_COPIES = 0x0040, /* Can do copies */
473     PAPI_PRINTER_COLLATE = 0x0080, /* Can collage copies */
474     PAPI_PRINTER_PUNCH = 0x0100, /* Can punch output */
475     PAPI_PRINTER_COVER = 0x0200, /* Can cover output */
476     PAPI_PRINTER_BIND = 0x0400, /* Can bind output */
477     PAPI_PRINTER_SORT = 0x0800, /* Can sort output */
478     PAPI_PRINTER_SMALL = 0x1000, /* Can do Letter/Legal/A4 */
479     PAPI_PRINTER_MEDIUM = 0x2000, /* Can do Tabloid/B/C/A3/A2 */
480     PAPI_PRINTER_LARGE = 0x4000, /* Can do D/E/A1/A0 */
481     PAPI_PRINTER_VARIABLE = 0x8000, /* Can do variable sizes */
482     PAPI_PRINTER_IMPLICIT = 0x10000, /* Implicit class */
483     PAPI_PRINTER_DEFAULT = 0x20000, /* Default printer on network */
484     PAPI_PRINTER_OPTIONS = 0xffff /* ~(CLASS | REMOTE | IMPLICIT) */
485 };
486

```

487 Chapter 4. Service API

488 4.1. papiServiceCreate

489 Description

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

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

497 Syntax

498

```
499 papi_status_t papiServiceCreate(  
500     papi_service_t*      handle,  
501     const char*          service_name,  
502     const char*          user_name,  
503     const char*          password,  
504     const int (*authCB)(papi_service_t svc),  
505     const papi_encryption_t encryption,  
506     void*                 app_data );  
507
```

508

509 Inputs

510

511 service_name

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

516 user_name

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

520 password

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

523 authCB

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

529 encryption

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

531 app_data

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

534

535 **Outputs**

536

537 handle

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

541

542 **Returns**

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

545 **Example**

546

```

547 #include "papi.h"
548
549 papi_status_t status;
550 papi_service_t handle = NULL;
551 const char* service_name = "ipp:/printserv:631";
552 const char* user_name = "pappy";
553 const char* password = "goober";
554 ...
555 status = papiServiceCreate(&handle,
556                             service_name,
557                             user_name,
558                             password,
559                             NULL,
560                             PAPI_ENCRYPT_IF_REQUESTED,
561                             NULL);
562
563 if (status != PAPI_OK)
564 {
565     /* handle the error */
566     fprintf(stderr, "papiServiceCreate failed: %s\n",
567             papiStatusString(status));
568     if (handle != NULL)
569     {
570         fprintf(stderr, "    details: %s\n",
571                 papiServiceGetStatusMessage(handle));
572     }
573     ...
574 }
575 ...
576 papiServiceDestroy(handle);
577

```

577

578 **See Also**

579 papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUsername,
580 papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB

581 **4.2. papiServiceDestroy**582 **Description**

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

588 **Syntax**

589

```
590 void papiServiceDestroy(  
591     papi_service_t handle );  
592
```

593

594 **Inputs**

595

596 handle

597 The print service handle to be destroyed.

598

599 **Outputs**

600 none

601 **Returns**

602 none

603 **Example**

604

```
605 #include "papi.h"  
606  
607 papi_status_t status;  
608 papi_service_t handle = NULL;  
609 const char* service_name = "ipp://printserv:631";  
610 const char* user_name = "pappy";  
611 const char* password = "goober";  
612 ...  
613 status = papiServiceCreate(&handle,  
614     service_name,  
615     user_name,  
616     password,  
617     NULL,  
618     PAPI_ENCRYPT_IF_REQUESTED,  
619     NULL);  
620  
621 if (status != PAPI_OK)  
622 {  
623     /* handle the error */  
624     ...  
625 }  
626 ...  
627 papiServiceDestroy(handle);
```

628

629 **See Also**

630 papiServiceCreate

631 **4.3. papiServiceSetUsername**632 **Description**

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

636 **Syntax**

637

```
638 papi_status_t papiServiceSetUsername(
639     papi_service_t handle,
640     const char* user_name );
641
```

642

643 **Inputs**

644

645 handle

646 Handle to the print service to update.

647 user_name

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

651

652 **Outputs**

653 handle is updated.

654 **Returns**

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

657 **Example**

658

```
659 #include "papi.h"
660
661 papi_status_t status;
662 papi_service_t handle = NULL;
663 const char* user_name = "pappy";
664 ...
665 status = papiServiceCreate(&handle,
666     NULL,
667     NULL,
668     NULL,
669     NULL,
670     PAPI_ENCRYPT_IF_REQUESTED,
671     NULL);
672
673 if (status != PAPI_OK)
674 {
675     /* handle the error */
676     ...
677 }
678
679 status = papiServiceSetUsername(handle, user_name);
680 if (status != PAPI_OK)
681 {
682     /* handle the error */
683     fprintf(stderr, "papiServiceSetUsername failed: %s\n",
684         papiServiceGetStatusMessage(handle));
```

```

684     ...
685 }
686 ...
687 papiServiceDestroy(handle);
688

```

689

690 **See Also**

691 papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

692 **4.4. papiServiceSetPassword**693 **Description**

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

697 **Syntax**

698

```

699 papi_status_t papiServiceSetPassword(
700     papi_service_t handle,
701     const char* password );
702

```

703

704 **Inputs**

705

706 handle

707 Handle to the print service to update.

708 password

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

710

711 **Outputs**

712 handle is updated.

713 **Returns**

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

716 **Example**

717

```

718 #include "papi.h"
719
720 papi_status_t status;
721 papi_service_t handle = NULL;
722 const char* password = "goober";
723 ...
724 status = papiServiceCreate(&handle,
725     NULL,
726     NULL,
727     NULL,
728     NULL,
729     PAPI_ENCRYPT_IF_REQUESTED,
730     NULL);
731
732 if (status != PAPI_OK)

```

```

732     {
733         /* handle the error */
734         ...
735     }
736
737     status = papiServiceSetPassword(handle, password);
738     if (status != PAPI_OK)
739     {
740         /* handle the error */
741         fprintf(stderr, "papiServiceSetPassword failed: %s\n",
742                papiServiceGetStatusMessage(handle));
743         ...
744     }
745     ...
746     papiServiceDestroy(handle);
747

```

748

749 **See Also**

750 papiServiceCreate, papiServiceSetUsername, papiServiceGetStatusMessage

751 4.5. papiServiceSetEncryption

752 **Description**

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

754 **Syntax**

755

```

756 papi_status_t papiServiceSetEncryption(
757     papi_service_t handle,
758     const papi_encryption_t encryption );
759

```

760

761 **Inputs**

762

763 handle

764 Handle to the print service to update.

765 encryption

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

767

768 **Outputs**

769 handle is updated.

770 **Returns**

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

773 **Example**

774

```

775 #include "papi.h"
776
777 papi_status_t status;
778 papi_service_t handle = NULL;
779 ...
780 status = papiServiceCreate(&handle,

```

```

781             NULL,
782             NULL,
783             NULL,
784             NULL,
785             PAPI_ENCRYPT_IF_REQUESTED,
786             NULL);
787
788     if (status != PAPI_OK)
789     {
790         /* handle the error */
791         ...
792     }
793
794     status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
795     if (status != PAPI_OK)
796     {
797         /* handle the error */
798         fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
799                 papiServiceGetStatusMessage(handle));
800         ...
801     }
802     ...
803     papiServiceDestroy(handle);

```

804

805 **See Also**

806 papiServiceCreate, papiServiceGetStatusMessage

807 **4.6. papiServiceSetAuthCB**808 **Description**

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

811 **Syntax**

812

```

813 papi_status_t papiServiceSetAuthCB(
814     papi_service_t handle,
815     const int (*authCB)(papi_service_t svc) );
816

```

817

818 **Inputs**

819

820 handle

821 Handle to the print service to update.

822 authCB

823 Points to a callback function to be used in authenticating the user to the print
824 service if no password was supplied (or user input is required). A NULL value
825 indicates that no callback should be made. The callback function should return
826 0 if the request is to be cancelled and non-zero if new authentication
827 information has been set.

828

829 **Outputs**

830 handle is updated.

831

Returns

832

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

833

834

Example

835

836

```

#include "papi.h"

extern int get_password(papi_service_t handle);
papi_status_t status;
papi_service_t handle = NULL;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_IF_REQUESTED,
                           NULL);

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

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

```

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

866

867

See Also

868

papiServiceCreate, papiServiceGetStatusMessage

4.7. papiServiceSetAppData

870

Description

871

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.

872

873

874

Syntax

875

876

```

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

```

877

878

879

880

881

Inputs

882

883

handle

884

Handle to the print service to update.

885 app_data

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

888

889 **Outputs**

890 handle is updated.

891 **Returns**

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

894 **Example**

895

```

896 #include "papi.h"
897
898 extern int get_password(papi_service_t handle);
899 papi_status_t status;
900 papi_service_t handle = NULL;
901 char* app_data = "some data";
902 ...
903 status = papiServiceCreate(&handle,
904                             NULL,
905                             NULL,
906                             NULL,
907                             NULL,
908                             PAPI_ENCRYPT_IF_REQUESTED,
909                             NULL);
910
911 if (status != PAPI_OK)
912 {
913     /* handle the error */
914     ...
915 }
916
917 status = papiServiceSetAppData(handle, app_data);
918 if (status != PAPI_OK)
919 {
920     /* handle the error */
921     fprintf(stderr, "papiServiceSetAppData failed: %s\n",
922            papiServiceGetStatusMessage(handle));
923     ...
924 }
925 ...
926 papiServiceDestroy(handle);

```

927

928 **See Also**

929 papiServiceCreate, papiServiceGetStatusMessage

930 **4.8. papiServiceGetServicename**

931 **Description**

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

933 **Syntax**

934

```

935 char* papiServiceGetServicename(
936     papi_service_t handle );
937

```

938

939 **Inputs**

940

941 handle

942 Handle to the print service.

943

944 **Outputs**

945 none

946 **Returns**

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

948 **Example**

949

```
950           #include "papi.h"
951
952           papi_status_t status;
953           papi_service_t handle = NULL;
954           char* service_name = NULL;
955           ...
956           service_name = papiServiceGetServicename(handle);
957           if (service_name != NULL)
958           {
959               /* use the returned name */
960               ...
961           }
962           ...
963           papiServiceDestroy(handle);
964
```

965

966 **See Also**

967 papiServiceCreate

968 **4.9. papiServiceGetUsername**

969 **Description**

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

971 **Syntax**

972

```
973           char* papiServiceGetUsername(
974               papi_service_t handle );
975
```

976

977 **Inputs**

978

979 handle

980 Handle to the print service.

981

982 **Outputs**

983 none

984 **Returns**

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

986 **Example**

987

```

988       #include "papi.h"
989
990       papi_status_t status;
991       papi_service_t handle = NULL;
992       char* user_name = NULL;
993       ...
994       user_name = papiServiceGetUsername(handle);
995       if (user_name != NULL)
996       {
997           /* use the returned name */
998           ...
999       }
1000       ...
1001       papiServiceDestroy(handle);
1002

```

1003

1004 **See Also**

1005 papiServiceCreate, papiServiceSetUsername

1006 **4.10. papiServiceGetPassword**1007 **Description**

1008 Get the user password associated with the print service handle.

1009 **Syntax**

1010

```

1011       char* papiServiceGetPassword(
1012           papi_service_t handle );
1013

```

1014

1015 **Inputs**

1016

1017 handle

1018 Handle to the print service.

1019

1020 **Outputs**

1021 none

1022 **Returns**

1023 A pointer to the password associated with the print service handle.

1024 **Example**

1025

```
1026     #include "papi.h"
1027
1028     papi_status_t status;
1029     papi_service_t handle = NULL;
1030     char* password = NULL;
1031     ...
1032     password = papiServiceGetPassword(handle);
1033     if (password != NULL)
1034     {
1035         /* use the returned password */
1036         ...
1037     }
1038     ...
1039     papiServiceDestroy(handle);
1040
```

1041

See Also

1042

papiServiceCreate, papiServiceSetPassword

1043

4.11. papiServiceGetEncryption

1044

Description

1045

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

1046

Syntax

1047

1048

```
1049     papi_encryption_t papiServiceGetEncryption(
1050         papi_service_t handle );
1051
```

1052

Inputs

1053

1054

handle

1055

Handle to the print service.

1056

1057

Outputs

1058

none

1059

Returns

1060

The type of encryption associated with the print service handle.

1061

Example

1062

1063

```
1064     #include "papi.h"
1065
1066     papi_status_t status;
1067     papi_service_t handle = NULL;
1068     papi_encryption_t encryption;
1069     ...
1070     encryption = papiServiceGetEncryption(handle);
1071     /* use the returned encryption value */
1072     ...
1073     papiServiceDestroy(handle);
1074
```

1075

1076 **See Also**
 1077 papiServiceCreate, papiServiceSetEncryption

1078 **4.12. papiServiceGetAppData**

1079 **Description**

1080 Get a pointer to the application-specific data associated with the print service
 1081 handle.

1082 **Syntax**

1083

```
1084           void* papiServiceGetAppData(  
1085                    papi_service_t handle );  
1086
```

1087

1088 **Inputs**

1089

1090 handle

1091 Handle to the print service.

1092

1093 **Outputs**

1094 none

1095 **Returns**

1096 A pointer to the application-specific data associated with the print service handle.

1097 **Example**

1098

```
1099           #include "papi.h"  
1100  
1101           papi_status_t status;  
1102           papi_service_t handle = NULL;  
1103           char* app_data = NULL;  
1104           ...  
1105           app_data = (char*)papiServiceGetAppData(handle);  
1106           if (app_data != NULL)  
1107           {  
1108                /* use the returned application data */  
1109                ...  
1110           }  
1111           ...  
1112           papiServiceDestroy(handle);  
1113
```

1114

1115 **See Also**

1116 papiServiceCreate, papiServiceSetAppData

1117 **4.13. papiServiceGetStatusMessage**

1118 **Description**

1119 Get the message associated with the status of the last operation performed. The
 1120 status message returned from this function may be more detailed than the status

1121 message returned from `papiStatusString` (if the print service supports returning
1122 more detailed error messages).

1123 The returned message will be localized in the language of the submitter of the
1124 original operation.

1125 **Syntax**

1126

```
1127 const char* papiServiceGetStatusMessage(  
1128     const papi_service_t handle );  
1129
```

1130

1131 **Inputs**

1132

1133 handle

1134 Handle to the print service.

1135

1136 **Outputs**

1137 none

1138 **Returns**

1139 Pointer to the message associated with the status of the last operation performed.

1140 **Example**

1141

```
1142 #include "papi.h"  
1143  
1144 papi_status_t status;  
1145 papi_service_t handle = NULL;  
1146 const char* user_name = "pappy";  
1147 ...  
1148 status = papiServiceCreate(&handle,  
1149                             NULL,  
1150                             NULL,  
1151                             NULL,  
1152                             NULL,  
1153                             PAPI_ENCRYPT_IF_REQUESTED,  
1154                             NULL);  
1155  
1156 if (status != PAPI_OK)  
1157 {  
1158     /* handle the error */  
1159     ...  
1160 }  
1161  
1162 status = papiServiceSetUsername(handle, user_name);  
1163 if (status != PAPI_OK)  
1164 {  
1165     /* handle the error */  
1166     fprintf(stderr, "papiServiceSetUsername failed: %s\n",  
1167             papiServiceGetStatusMessage(handle));  
1168     ...  
1169 }  
1170 ...  
1171 papiServiceDestroy(handle);
```

1172

1173 **See Also**

1174 `papiStatusString`

1175 Chapter 5. Printer API

1176 5.1. Usage

1177 The papiPrinterQuery function queries all/some of the attributes of a printer
1178 object. It returns a list of printer attributes. A successful call to papiPrinterQuery is
1179 typically followed by code which examines and processes the returned attributes.
1180 The using program would then call papiPrinterFree to delete the returned results.

1181 Printers can be found via calls to papiPrintersList. A successful call to
1182 papiPrintersList is typically followed by code to iterate through the list of returned
1183 printers, possibly querying each (papiPrinterQuery) for further information (e.g. to
1184 restrict what printers get displayed for a particular user/request). The using
1185 program would then call papiPrinterListFree to free the returned results.

1186 5.2. papiPrintersList

1187 Description

1188 List all printers known by the print service which match the specified filter.

1189 Depending on the functionality of the target service's "printer directory", the
1190 returned list may be limited to only printers managed by a particular server or it
1191 may include printers managed by other servers.

1192 Syntax

```
1194 papi_status_t papiPrintersList(  
1195             papi_service_t   handle,  
1196             const char*      requested_attrs[],  
1197             const papi_filter_t* filter,  
1198             papi_printer_t** printers );  
1199
```

1201 Inputs

1203 handle

1204 Handle to the print service to use.

1205 requested_attrs

1206 (optional) NULL terminated array of attributes to be queried. If NULL is
1207 passed then all attributes are queried. (NOTE: The printer may return more
1208 attributes than you requested. This is merely an advisory request that may
1209 reduce the amount of data returned if the printer/server supports it.)

1210 filter

1211 (optional) Pointer to a filter to limit the number of printers returned on the list
1212 request. See Section 3.9 for details. If NULL is passed then all known printers
1213 are listed.

1214

1215 **Outputs**

1216

1217 printers

1218 List of printer objects that matched the filter criteria.

1219

1220 **Returns**

1221 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1222 value is returned.

1223 **Example**

1224

```

1225     #include "papi.h"
1226
1227     int i;
1228     papi_status_t status;
1229     papi_service_t handle = NULL;
1230     const char* service_name = "ipp://printserv:631";
1231     const char* user_name = "pappy";
1232     const char* password = "goober";
1233     const char* req_attrs[] =
1234     {
1235         "printer-name",
1236         "printer-location",
1237         NULL
1238     };
1239     const papi_filter_t filter =
1240         PAPI_PRINTER_BW | PAPI_PRINTER_DUPLEX;
1241     papi_printer_t* printers = NULL;
1242     ...
1243     status = papiServiceCreate(&handle,
1244                               service_name,
1245                               user_name,
1246                               password,
1247                               NULL,
1248                               PAPI_ENCRYPT_IF_REQUESTED,
1249                               NULL);
1250
1251     if (status != PAPI_OK)
1252     {
1253         /* handle the error */
1254         ...
1255     }
1256
1257     status = papiPrinterList(handle,
1258                             req_attrs,
1259                             filter,
1260                             &printers);
1261
1262     if (status != PAPI_OK)
1263     {
1264         /* handle the error */
1265         fprintf(stderr, "papiPrinterList failed: %s\n",
1266                papiServiceGetStatusMessage(handle));
1267         ...
1268     }
1269
1270     if (printers != NULL)
1271     {
1272         for (i=0; printers[i] != NULL; i++)
1273         {
1274             /* process the printer object */
1275             ...
1276         }
1277         papiPrinterListFree(printers);
1278     }
1279
1280     papiServiceDestroy(handle);

```

1280

1281 **See Also**

1282 papiPrinterListFree, papiPrinterQuery

1283 **5.3. papiPrinterQuery**1284 **Description**

1285 Queries some or all the attributes of the specified printer object. This includes
 1286 attributes representing the capabilities of the printer, which the caller may use to
 1287 determine which print options to present to the user. How the attributes are
 1288 obtained (e.g. from a static database, from a dialog with the hardware, from a dialog
 1289 with a driver, etc.) is up to the implementer of the API and is beyond the scope of
 1290 this standard.

1291 This optionally includes "context" information which specifies job attributes in the
 1292 context of which the capabilities information is to be constructed.

1293 **Semantics Reference**

1294 Get-Printer-Attributes in [RFC2911], section 3.2.5

1295 **Syntax**

1296

```

1297 papi_status_t papiPrinterQuery(
1298             papi_service_t   handle,
1299             const char*       name,
1300             const char*       requested_attrs[],
1301             const papi_attribute_t** job_attrs,
1302             papi_printer_t*   printer );
1303

```

1304

1305 **Inputs**

1306

1307 handle

1308 Handle to the print service to use.

1309 name

1310 The name or URI of the printer to query.

1311 requested_attrs

1312 (optional) NULL terminated array of attributes to be queried. If NULL is
 1313 passed then all attributes are queried. (NOTE: The printer may return more
 1314 attributes than you requested. This is merely an advisory request that may
 1315 reduce the amount of data returned if the printer/server supports it.)

1316 job_attrs

1317 (optional) NULL terminated array of job attributes in the context of which the
 1318 capabilities information is to be constructed. In other words, the returned
 1319 printer attributes represent the capabilities of the printer given that these
 1320 specified job attributes are requested. This allows for more accurate
 1321 information to be retrieved by the caller for a specific job (e.g. "if the job is
 1322 printed on A4 size media then duplex output is not available"). If NULL is
 1323 passed then the full capabilities of the printer are queried.

1324 Support for this argument is optional. If the underlying print system does not
 1325 have access to capabilities information bound by job context, then this

1326 argument may be ignored. But if the calling application will be using the
 1327 returned information to build print job data, then it is always advisable to
 1328 specify the job context attributes. The more context information provided, the
 1329 more accurate capabilities information is likely to be returned from the print
 1330 system.

1331

1332 **Outputs**

1333

1334 printer

1335 Pointer to a printer object containing the requested attributes.

1336

1337 **Returns**

1338 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1339 value is returned.

1340 **Example**

1341

```

1342 #include "papi.h"
1343
1344 papi_status_t status;
1345 papi_service_t handle = NULL;
1346 const char* service_name = "ipp://printserv:631";
1347 const char* user_name = "pappy";
1348 const char* password = "goober";
1349 const char* printer_name = "my-printer";
1350 const char* req_attrs[] =
1351 {
1352     "printer-name",
1353     "printer-location",
1354     "printer-state",
1355     "printer-state-reasons",
1356     "printer-state-message",
1357     NULL
1358 };
1359
1360 papi_attribute_t** job_attrs = NULL;
1361 papi_printer_t printer = NULL;
1362 ...
1363 status = papiServiceCreate(&handle,
1364                             service_name,
1365                             user_name,
1366                             password,
1367                             NULL,
1368                             PAPI_ENCRYPT_IF_REQUESTED,
1369                             NULL);
1370
1371 if (status != PAPI_OK)
1372 {
1373     /* handle the error */
1374     ...
1375 }
1376
1377 papiAttributeListAddString(&job_attrs,
1378                             PAPI_EXCL,
1379                             "media",
1380                             "legal");
1381
1382 status = papiPrinterQuery(handle,
1383                             printer_name,
1384                             req_attrs,
1385                             job_attrs,
1386                             &printer);
1387
1388 if (status != PAPI_OK)
1389 {
1390     /* handle the error */
1391     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1392             papiServiceGetStatusMessage(handle));
1393     ...
1394 }
1395
1396 if (printer != NULL)
1397 {
  
```



```

1395     /* process the printer object */
1396     ...
1397     papiPrinterFree(printer);
1398 }
1399
1400 papiAttributeListFree(job_attrs);
1401 papiServiceDestroy(handle);
1402

```

1403

1404

See Also

1405

papiPrinterList, papiPrinterFree, papiPrinterModify

1406

5.4. papiPrinterModify

1407

Description

1408

1409

1410

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.

1411

Semantics Reference

1412

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

1413

Syntax

1414

1415

1416

1417

1418

1419

1420

```

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

```

1421

1422

Inputs

1423

1424

handle

1425

Handle to the print service to use.

1426

printer_name

1427

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

1428

attrs

1429

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

1430

1431

Outputs

1432

1433

printer

1434

The modified printer object.

1435

1436

Returns

1437

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

1438

1439

Example

1440

```

1441 #include "papi.h"
1442
1443 papi_status_t status;
1444 papi_service_t handle = NULL;
1445 const char* printer_name = "my-printer";
1446 papi_printer_t printer = NULL;
1447 papi_attribute_t** attrs = NULL;
1448 ...
1449 status = papiServiceCreate(&handle,
1450                          NULL,
1451                          NULL,
1452                          NULL,
1453                          NULL,
1454                          PAPI_ENCRYPT_NEVER,
1455                          NULL);
1456
1457 if (status != PAPI_OK)
1458 {
1459     /* handle the error */
1460     ...
1461 }
1462
1463 papiAttributeListAddString(&attrs,
1464                          PAPI_EXCL,
1465                          "printer-location",
1466                          "Bldg 17/Room 234");
1467
1468 status = papiPrinterModify(handle,
1469                          printer_name,
1470                          attrs,
1471                          &printer);
1472
1473 if (status != PAPI_OK)
1474 {
1475     /* handle the error */
1476     fprintf(stderr, "papiPrinterModify failed: %s\n",
1477            papiServiceGetStatusMessage(handle));
1478     ...
1479 }
1480
1481 if (printer != NULL)
1482 {
1483     /* process the printer */
1484     ...
1485     papiPrinterFree(printer);
1486 }
1487
1488 papiServiceDestroy(handle);

```

1488

1489

See Also

1490

papiPrinterQuery, papiPrinterFree

1491

5.5. papiPrinterPause

1492

Description

1493

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

1494

1495

1496

1497

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

1498

1499

Semantics Reference

1500

Pause-Printer in [RFC2911], section 3.2.7

1501 **Syntax**

1502

```
1503     papi_status_t papiPrinterPause(
1504         papi_service_t     handle,
1505         const char*         name,
1506         const char*         message );
1507
```

1508

1509 **Inputs**

1510

1511 handle

1512 Handle to the print service to use.

1513 name

1514 The name or URI of the printer to operate on.

1515 message

1516 (optional) An explanatory message to be associated with the paused printer.
1517 This message may be ignored if the underlying print system does not support
1518 associating a message with a paused printer.

1519

1520 **Outputs**

1521 none

1522 **Returns**

1523 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1524 value is returned.

1525 **Example**

1526

```
1527     #include "papi.h"
1528
1529     papi_status_t status;
1530     papi_service_t handle = NULL;
1531     const char* service_name = "ipp://printserv:631";
1532     const char* user_name = "pappy";
1533     const char* password = "goober";
1534     const char* printer_name = "my-printer";
1535     ...
1536     status = papiServiceCreate(&handle,
1537         service_name,
1538         user_name,
1539         password,
1540         NULL,
1541         PAPI_ENCRYPT_IF_REQUESTED,
1542         NULL);
1543
1544     if (status != PAPI_OK)
1545     {
1546         /* handle the error */
1547         ...
1548     }
1549
1550     status = papiPrinterPause(handle, printer_name, NULL);
1551     if (status != PAPI_OK)
1552     {
1553         /* handle the error */
1554         fprintf(stderr, "papiPrinterPause failed: %s\n",
1555             papiServiceGetStatusMessage(handle));
1556         ...
1557     }
```

```

1556     }
1557     ...
1558     papiServiceDestroy(handle);
1559

```

1560

1561 **See Also**

1562 papiPrinterResume

1563 **5.6. papiPrinterResume**1564 **Description**

1565 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the
 1566 effects of papiPrinterPause). This operation is optional and may not be supported
 1567 by all printers/servers, but it must be supported if papiPrinterPause is supported.

1568 **Semantics Reference**

1569 Resume-Printer in [RFC2911], section 3.2.8

1570 **Syntax**

1571

```

1572 papi_status_t papiPrinterResume(
1573             papi_service_t   handle,
1574             const char*      name );
1575

```

1576

1577 **Inputs**

1578

1579 handle

1580 Handle to the print service to use.

1581 name

1582 The name or URI of the printer to operate on.

1583

1584 **Outputs**

1585 none

1586 **Returns**

1587 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1588 value is returned.

1589 **Example**

1590

```

1591 #include "papi.h"
1592
1593 papi_status_t status;
1594 papi_service_t handle = NULL;
1595 const char* service_name = "ipp://printserv:631";
1596 const char* user_name = "pappy";
1597 const char* password = "goober";
1598 const char* printer_name = "my-printer";
1599 ...
1600 status = papiServiceCreate(&handle,

```

```

1601         service_name,
1602         user_name,
1603         password,
1604         NULL,
1605         PAPI_ENCRYPT_IF_REQUESTED,
1606         NULL);
1607     if (status != PAPI_OK)
1608     {
1609         /* handle the error */
1610         ...
1611     }
1612
1613     status = papiPrinterPause(handle, printer_name);
1614     if (status != PAPI_OK)
1615     {
1616         /* handle the error */
1617         fprintf(stderr, "papiPrinterPause failed: %s\n",
1618                papiServiceGetStatusMessage(handle));
1619         ...
1620     }
1621     ...
1622     status = papiPrinterResume(handle, printer_name);
1623     if (status != PAPI_OK)
1624     {
1625         /* handle the error */
1626         fprintf(stderr, "papiPrinterResume failed: %s\n",
1627                papiServiceGetStatusMessage(handle));
1628         ...
1629     }
1630
1631     papiServiceDestroy(handle);
1632

```

1633

1634

See Also

1635

papiPrinterPause

1636

5.7. papiPrinterPurgeJobs

1637

Description

1638

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).

1639

1640

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

1641

Semantics Reference

1642

Purge-Jobs in [RFC2911], section 3.2.9

1643

Syntax

1644

1645

```

papi_status_t papiPrinterPurgeJobs(
1646     papi_service_t    handle,
1647     const char*       name,
1648     papi_job_t**      result);
1649

```

1650

1651

Inputs

1652

1653

handle

1654

Handle to the print service to use.

1655

name

1656

The name or URI of the printer to operate on.

1657

1658

Outputs

1659

1660 result

1661

1662

1663

1664

1665

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

1666 name

1667

1668

The name or URI of the printer to operate on.

1669

Returns

1670

1671

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

1672

Example

1673

1674

1675

1676

1677

1678

1679

1680

1681

1682

1683

1684

1685

1686

1687

1688

1689

1690

1691

1692

1693

1694

1695

1696

1697

1698

1699

1700

1701

1702

1703

1704

1705

```
#include "papi.h"

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

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

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

papiServiceDestroy(handle);
```

1707

1708

See Also

1709

papiJobCancel

5.8. papiPrinterListJobs

1711

Description

1712

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

1713 **Semantics Reference**

1714 Get-Jobs in [RFC2911], section 3.2.6

1715 **Syntax**

1716

```

1717 papi_status_t papiPrinterListJobs(
1718             papi_service_t   handle,
1719             const char*       printer,
1720             const char*       requested_attrs[],
1721             const int          type_mask,
1722             const int          max_num_jobs,
1723             papi_job_t**      jobs );
1724

```

1725

1726 **Inputs**

1727

1728 handle

1729 Handle to the print service to use.

1730 requested_attrs

1731 (optional) NULL terminated array of attributes to be queried. If NULL is
 1732 passed then all available attributes are queried. (NOTE: The printer may return
 1733 more attributes than you requested. This is merely an advisory request that
 1734 may reduce the amount of data returned if the printer/server supports it.)

1735 type_mask

1736 A bit mask which determines what jobs will get returned. The following
 1737 constants can be bitwise-OR-ed together to select which types of jobs to list:

```

1738 #define PAPI_LIST_JOBS_OTHERS      0x0001 /* return jobs other than
1739 those submitted by the
1740 user name assoc with
1741 the handle */
1742 #define PAPI_LIST_JOBS_COMPLETED  0x0002 /* return completed jobs */
1743 #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
1744 jobs */
1745 #define PAPI_LIST_JOBS_ALL        0xFFFF /* return all jobs */
1746

```

1747

1748 max_num_jobs

1749 Limit to the number of jobs returned. If 0 is passed, then there is no limit on
 1750 the number of jobs which may be returned.

1751

1752 **Outputs**

1753

1754 jobs

1755 List of job objects returned.

1756

1757

Returns

1758

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

1759

1760

Example

1761

```

1762 #include "papi.h"
1763
1764 int i;
1765 papi_status_t status;
1766 papi_service_t handle = NULL;
1767 const char* printer_name = "my-printer";
1768 papi_job_t* jobs = NULL;
1769 const char* job_attrs[] =
1770 {
1771     "job-id",
1772     "job-name",
1773     "job-originating-user-name",
1774     "job-state",
1775     "job-state-reasons",
1776     NULL
1777 };
1778 ...
1779 status = papiServiceCreate(&handle,
1780     NULL,
1781     NULL,
1782     NULL,
1783     NULL,
1784     PAPI_ENCRYPT_NEVER,
1785     NULL);
1786
1787 if (status != PAPI_OK)
1788 {
1789     /* handle the error */
1790     ...
1791 }
1792
1793 status = papiPrinterListJobs(handle,
1794     printer_name,
1795     job_attrs,
1796     PAPI_LIST_JOBS_ALL,
1797     0,
1798     &jobs);
1799
1800 if (status != PAPI_OK)
1801 {
1802     /* handle the error */
1803     fprintf(stderr, "papiPrinterListJobs failed: %s\n",
1804         papiServiceGetStatusMessage(handle));
1805     ...
1806 }
1807
1808 if (jobs != NULL)
1809 {
1810     for(i=0; jobs[i] != NULL; i++)
1811     {
1812         /* process the job */
1813         ...
1814     }
1815     papiJobListFree(jobs);
1816 }
1817
1818 papiServiceDestroy(handle);

```

1818

See Also

1819

papiJobQuery, papiJobListFree

1820

5.9. papiPrinterGetAttributeList

1821

Description

1822

Get the attribute list associated with a printer object.

1823

1824 This function retrieves an attribute list from a printer object returned in a previous
 1825 call. Printer objects are returned as the result of operations performed by
 1826 `papiPrintersList`, `papiPrinterQuery`, and `papiPrinterModify`.

1827 **Syntax**

1828

```
1829 papi_attribute_t** papiPrinterGetAttributeList(  
1830     papi_printer_t printer );  
1831
```

1832

1833 **Inputs**

1834

1835 printer

Handle of the printer object.

1837

1838 **Outputs**

1839 none

1840 **Returns**

1841 Pointer to the attribute list associated with the printer object.

1842 **Example**

1843

```
1844 #include "papi.h"  
1845  
1846 papi_status_t status;  
1847 papi_service_t handle = NULL;  
1848 const char* printer_name = "my-printer";  
1849 papi_printer_t printer = NULL;  
1850 papi_attribute_list* attrs = NULL;  
1851 ...  
1852 status = papiServiceCreate(&handle,  
1853     NULL,  
1854     NULL,  
1855     NULL,  
1856     NULL,  
1857     PAPI_ENCRYPT_NEVER,  
1858     NULL);  
1859  
1860 if (status != PAPI_OK)  
1861 {  
1862     /* handle the error */  
1863     ...  
1864 }  
1865  
1866 status = papiPrinterQuery(handle,  
1867     printer_name,  
1868     NULL,  
1869     &printer);  
1870  
1871 if (status != PAPI_OK)  
1872 {  
1873     /* handle the error */  
1874     fprintf(stderr, "papiPrinterQuery failed: %s\n",  
1875         papiServiceGetStatusMessage(handle));  
1876     ...  
1877 }  
1878  
1879 if (printer != NULL)  
1880 {  
1881     /* process the printer object */  
1882     attrs = papiPrinterGetAttributeList(printer);  
1883     ...  
1884     papiPrinterFree(printer);  
1885 }  
1886  
1887 papiServiceDestroy(handle);
```

1886

1887

1888

See Also

1889

papiPrintersList, papiPrinterQuery

1890

5.10. papiPrinterFree

1891

Description

1892

Free a printer object.

1893

Syntax

1894

1895

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

1896

1897

1898

1899

Inputs

1900

1901

printer

1902

Handle of the printer object to free.

1903

1904

Outputs

1905

none

1906

Returns

1907

none

1908

Example

1909

1910

```
#include "papi.h"

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

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

status = papiPrinterQuery(handle,
    printer_name,
    NULL,
    &printer);

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

1911

1912

1913

1914

1915

1916

1917

1918

1919

1920

1921

1922

1923

1924

1925

1926

1927

1928

1929

1930

1931

1932

1933

1934

1935

1936

1937

1938

1939

```

1940     }
1941
1942     if (printer != NULL)
1943     {
1944         /* process the printer object */
1945         ...
1946         papiPrinterFree(printer);
1947     }
1948
1949     papiServiceDestroy(handle);
1950

```

1951

1952 **See Also**1953 `papiPrinterQuery`1954 **5.11. papiPrinterListFree**1955 **Description**

1956 Free a list of printer objects.

1957 **Syntax**

1958

```

1959 void papiPrinterListFree(
1960     papi_printer_t*   printers );
1961

```

1962

1963 **Inputs**

1964

1965 `printers`

1966 Pointer to the printer object list to free.

1967

1968 **Outputs**

1969 none

1970 **Returns**

1971 none

1972 **Example**

1973

```

1974 #include "papi.h"
1975
1976 papi_status_t status;
1977 papi_service_t handle = NULL;
1978 const char* printer_name = "my-printer";
1979 papi_printer_t* printers = NULL;
1980 ...
1981 status = papiServiceCreate(&handle,
1982     NULL,
1983     NULL,
1984     NULL,
1985     NULL,
1986     PAPI_ENCRYPT_NEVER,
1987     NULL);
1988
1989 if (status != PAPI_OK)
1990 {
1991     /* handle the error */
1992     ...
1993 }

```

```
1994     status = papiPrinterList(handle,  
1995                               NULL,  
1996                               NULL,  
1997                               &printers);  
1998     if (status != PAPI_OK)  
1999     {  
2000         /* handle the error */  
2001         fprintf(stderr, "papiPrinterList failed: %s\n",  
2002                papiServiceGetStatusMessage(handle));  
2003         ...  
2004     }  
2005  
2006     if (printers != NULL)  
2007     {  
2008         /* process the printer objects */  
2009         ...  
2010         papiPrinterListFree(printers);  
2011     }  
2012  
2013     papiServiceDestroy(handle);  
2014
```

2015

2016

See Also

2017

papiPrinterList

2018 **Chapter 6. Attributes API**

2019 **6.1. papiAttributeListAdd**

2020 **Description**

2021 Add an attribute/value to an attribute list. Depending on the `add_flags`, this may
2022 also be used to add values to an existing multivalued attribute. Memory is allocated
2023 and copies of the input arguments are created. It is the caller's responsibility to call
2024 `papiAttributeListFree` when done with the attribute list.

2025 This function is equivalent to the `papiAttributeListAddString`,
2026 `papiAttributeListAddInteger`, etc. functions defined later in this chapter.

2027 **Syntax**

2028

```
2029 papi_status_t papiAttributeListAdd(  
2030     papi_attribute_t*** attrs,  
2031     const int add_flags,  
2032     const char* name,  
2033     const papi_attribute_value_type_t type,  
2034     const papi_attribute_value_t* value );  
2035
```

2036

2037 **Inputs**

2038

2039 `attrs`

2040 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
2041 `NULL` then this function will allocate the attribute list.

2042 `add_flags`

2043 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
2044 that indicates how to handle the request.

2045 `name`

2046 Points to the name of the attribute to add.

2047 `type`

2048 The type of values for this attribute.

2049 `value`

2050 Points to the attribute value to be added.

2051

2052 **Outputs**

2053

2054 `attrs`

2055 The attribute list is updated.

2056

2057

Returns

2058

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

2059

2060

Example

2061

2062

```
#include "papi.h"
2063
2064 papi_attribute_t** attrs = NULL;
2065 ...
2066 papiAttributeListAdd(&attrs,
2067                     PAPI_EXCL,
2068                     "job-name",
2069                     PAPI_STRING,
2070                     "My job" );
2071 ...
2072 papiAttributeListFree(attrs);
2073
```

2074

2075

See Also

2076

papiAttributeListFree, papiAttributeListAddString, papiAttributeListAddInteger,
 2077 papiAttributeListAddBoolean, papiAttributeListAddRange,
 2078 papiAttributeListAddResolution, papiAttributeListAddDatetime

2079

6.2. papiAttributeListAddString

2080

Description

2081

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

2082

2083

2084

2085

Syntax

2086

2087

```
papi_status_t papiAttributeListAddString(
2088     papi_attribute_t*** attrs,
2089     const int add_flags,
2090     const char* name,
2091     const char* value );
2092
```

2093

2094

Inputs

2095

2096 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.

2097

2098

2099 add_flags

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

2100

2101

2102 name
 2103 Points to the name of the attribute to add.

2104 value
 2105 The value to be added.

2106

2107 **Outputs**

2108

2109 attrs
 2110 The attribute list is updated.

2111

2112 **Returns**

2113 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2114 value is returned.

2115 **Example**

2116

```

2117 #include "papi.h"
2118
2119 papi_attribute_t** attrs = NULL;
2120 ...
2121 papiAttributeListAddString(&attrs,
2122                             PAPI_EXCL,
2123                             "job-name",
2124                             "My job" );
2125 ...
2126 papiAttributeListFree(attrs);
2127
```

2128

2129 **See Also**

2130 papiAttributeListFree, papiAttributeListAdd

2131 **6.3. papiAttributeListAddInteger**

2132 **Description**

2133 Add an integer-valued attribute to an attribute list. Depending on the add_flags,
 2134 this may also be used to add values to an existing multivalued attribute. Memory is
 2135 allocated and copies of the input arguments are created. It is the caller's
 2136 responsibility to call papiAttributeListFree when done with the attribute list.

2137 **Syntax**

2138

```

2139 papi_status_t papiAttributeListAddInteger(
2140     papi_attribute_t*** attrs,
2141     const int add_flags,
2142     const char* name,
2143     const int value );
2144
```

2145

2146 **Inputs**

2147

2148 `attrs`

2149 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
 2150 `NULL` then this function will allocate the attribute list.

2151 `add_flags`

2152 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 2153 that indicates how to handle the request.

2154 `name`

2155 Points to the name of the attribute to add.

2156 `value`

2157 The value to be added.

2158

2159 **Outputs**

2160

2161 `attrs`

2162 The attribute list is updated.

2163

2164 **Returns**

2165 If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate failure
 2166 value is returned.

2167 **Example**

2168

```

2169 #include "papi.h"
2170
2171 papi_attribute_t** attrs = NULL;
2172 ...
2173 papiAttributeListAddInteger(&attrs,
2174                             PAPI_EXCL,
2175                             "copies",
2176                             3 );
2177 ...
2178 papiAttributeListFree(attrs);
2179

```

2180

2181 **See Also**

2182 `papiAttributeListFree`, `papiAttributeListAdd`

2183 **6.4. papiAttributeListAddBoolean**2184 **Description**

2185 Add a boolean-valued attribute to an attribute list. Depending on the `add_flags`,
 2186 this may also be used to add values to an existing multivalued attribute. Memory is
 2187 allocated and copies of the input arguments are created. It is the caller's
 2188 responsibility to call `papiAttributeListFree` when done with the attribute list.

2189 **Syntax**

2190

```

2191     papi_status_t papiAttributeListAddBoolean(
2192         papi_attribute_t*** attrs,
2193         const int add_flags,
2194         const char* name,
2195         const char value );
2196 
```

2197

2198 **Inputs**

2199

2200 attrs

2201 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2202 NULL then this function will allocate the attribute list.

2203 add_flags

2204 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2205 that indicates how to handle the request.

2206 name

2207 Points to the name of the attribute to add.

2208 value

2209 The value (PAPI_FALSE or PAPI_TRUE) to be added.

2210

2211 **Outputs**

2212

2213 attrs

2214 The attribute list is updated.

2215

2216 **Returns**

2217 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2218 value is returned.

2219 **Example**

2220

```

2221     #include "papi.h"
2222
2223     papi_attribute_t** attrs = NULL;
2224     ...
2225     papiAttributeListAddBoolean(&attrs,
2226         PAPI_EXCL,
2227         "color-supported",
2228         PAPI_TRUE );
2229     ...
2230     papiAttributeListFree(attrs);
2231 
```

2232

2233 **See Also**
 2234 papiAttributeListFree, papiAttributeListAdd

2235 **6.5. papiAttributeListAddRange**

2236 **Description**

2237 Add a range-valued attribute to an attribute list. Depending on the `add_flags`, this
 2238 may also be used to add values to an existing multivalued attribute. Memory is
 2239 allocated and copies of the input arguments are created. It is the caller's
 2240 responsibility to call `papiAttributeListFree` when done with the attribute list.

2241 **Syntax**

2242

```
2243 papi_status_t papiAttributeListAddRange(  
2244     papi_attribute_t*** attrs,  
2245     const int add_flags,  
2246     const char* name,  
2247     const int lower,  
2248     const int upper );  
2249
```

2250

2251 **Inputs**

2252

2253 `attrs`

2254 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
 2255 `NULL` then this function will allocate the attribute list.

2256 `add_flags`

2257 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 2258 that indicates how to handle the request.

2259 `name`

2260 Points to the name of the attribute to add.

2261 `lower`

2262 The lower range value. This value must be less than or equal to the upper
 2263 range value.

2264 `upper`

2265 The upper range value. This value must be greater than or equal to the lower
 2266 range value.

2267

2268 **Outputs**

2269

2270 `attrs`

2271 The attribute list is updated.

2272

2273

Returns

2274

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

2275

2276

Example

2277

2278

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

2279

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

2280

```
...
```

2281

```
papiAttributeListAddRange (&attrs,
```

2282

```
    PAPI_EXCL,
```

2283

```
    "job-k-octets-supported",
```

2284

```
    1,
```

2285

```
    100000 );
```

2286

```
...
```

2287

```
papiAttributeListFree (attrs);
```

2288

2289

2290

2291

See Also

2292

papiAttributeListFree

2293

6.6. papiAttributeListAddResolution

2294

Description

2295

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.

2296

2297

2298

2299

Syntax

2300

2301

```
papi_status_t papiAttributeListAddResolution (
```

2302

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

2303

```
    const int add_flags,
```

2304

```
    const char* name,
```

2305

```
    const int xres,
```

2306

```
    const int yres,
```

2307

```
    const papi_res_t units );
```

2308

2309

2310

Inputs

2311

2312

attrs

2313

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.

2314

2315

add_flags

2316

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

2317

2318 name
2319 Points to the name of the attribute to add.

2320 xres
2321 The X-axis resolution value.

2322 yres
2323 The Y-axis resolution value.

2324 units
2325 The units of the resolution values provided.

2326

2327 **Outputs**

2328

2329 attrs
2330 The attribute list is updated.

2331

2332 **Returns**

2333 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2334 value is returned.

2335 **Example**

2336

```
2337 #include "papi.h"  
2338  
2339 papi_attribute_t** attrs = NULL;  
2340 ...  
2341 papiAttributeListAddResolution(&attrs,  
2342                               PAPI_EXCL,  
2343                               "printer-resolution",  
2344                               300,  
2345                               300,  
2346                               PAPI_RES_PER_INCH );  
2347 ...  
2348 papiAttributeListFree(attrs);  
2349
```

2350

2351 **See Also**

2352 papiAttributeListFree

2353 **6.7. papiAttributeListAddDatetime**

2354 **Description**

2355 Add a date/time-valued attribute to an attribute list. Depending on the add_flags,
2356 this may also be used to add values to an existing multivalued attribute. Memory is
2357 allocated and copies of the input arguments are created. It is the caller's
2358 responsibility to call papiAttributeListFree when done with the attribute list.

2359 **Syntax**

2360

```

2361     papi_status_t papiAttributeListAddDatetime(
2362         papi_attribute_t*** attrs,
2363         const int add_flags,
2364         const char* name,
2365         const time_t date_time );
2366

```

2367

2368 **Inputs**

2369

2370 attrs

2371 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
 2372 NULL then this function will allocate the attribute list.

2373 add_flags

2374 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2375 that indicates how to handle the request.

2376 name

2377 Points to the name of the attribute to add.

2378 date_time

2379 The date/time value.

2380

2381 **Outputs**

2382

2383 attrs

2384 The attribute list is updated.

2385

2386 **Returns**

2387 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2388 value is returned.

2389 **Example**

2390

```

2391     #include "papi.h"
2392
2393     papi_attribute_t** attrs = NULL;
2394     time_t date_time
2395     ...
2396     time(&date_time);
2397     papiAttributeListAddDatetime(&attrs,
2398         PAPI_EXCL,
2399         "date-time-at-creation",
2400         date_time );
2401     ...
2402     papiAttributeListFree(attrs);
2403

```

2404

2405 **See Also**

2406 `papiAttributeListFree`

2407 **6.8. papiAttributeListAddCollection**

2408 **Description**

2409 Add a collection-valued attribute to an attribute list. Depending on the `add_flags`,
2410 this may also be used to add values to an existing multivalued attribute. Memory is
2411 allocated and copies of the input arguments are created. It is the caller's
2412 responsibility to call `papiAttributeListFree` when done with the attribute list.

2413 **Syntax**

2414

```
2415 papi_status_t papiAttributeListAddCollection(  
2416     papi_attribute_t*** attrs,  
2417     const int add_flags,  
2418     const char* name,  
2419     const papi_attribute_t** collection );  
2420
```

2421

2422 **Inputs**

2423

2424 `attrs`

2425 Points to an attribute list. `attrs` equal to `NULL` is a bad argument, but if `*attrs` is
2426 `NULL` then this function will allocate the attribute list.

2427 `add_flags`

2428 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
2429 that indicates how to handle the request.

2430 `name`

2431 Points to the name of the attribute to add.

2432 `collection`

2433 The collection value.

2434

2435 **Outputs**

2436

2437 `attrs`

2438 The attribute list is updated.

2439

2440 **Returns**

2441 If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate failure
2442 value is returned.

2443

Example

2444

2445

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

2446

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

2447

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

2448

```
...
```

2449

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

2450

```
papiAttributeListAddString(&collection,
```

2451

```
    PAPI_EXCL,
```

2452

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

2453

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

2454

```
papiAttributeListAddString(&collection,
```

2455

```
    PAPI_EXCL,
```

2456

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

2457

```
    "stationery");
```

2458

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

2459

```
papiAttributeListAddCollection(&attrs,
```

2460

```
    PAPI_EXCL,
```

2461

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

2462

```
    collection );
```

2463

```
...
```

2464

```
papiAttributeListFree(collection);
```

2465

```
papiAttributeListFree(attrs);
```

2466

2467

2468

2469

See Also

2470

papiAttributeListFree

2471

2472

6.9. papiAttributeDelete

2473

Description

2474

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

2475

2476

Syntax

2477

2478

```
papi_status_t papiAttributeDelete(
```

2479

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

2480

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

2481

2482

2483

Inputs

2484

2485 attrs

Points to an attribute list.

2486

2487 name

Points to the name of the attribute to delete.

2488

2489

2490

Outputs

2491

2492 attrs

2493 The attribute list is updated.

2494

2495 **Returns**

2496 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2497 value is returned.

2498 **Example**

2499

```
2500 #include "papi.h"
2501
2502 papi_attribute_t** attrs = NULL;
2503 ...
2504 papiAttributeDelete(&attrs,
2505                    "copies" );
2506 ...
2507
```

2508

2509 **See Also**

2510 papiAttributeListFree

2511 **6.10. papiAttributeListGetValue**

2512 **Description**

2513 Get an attribute's value from an attribute list.

2514 This function is equivalent to the papiAttributeListGetString,
2515 papiAttributeListGetInteger, etc. functions defined later in this chapter.

2516 **Syntax**

2517

```
2518 papi_status_t papiAttributeListGetValue(
2519             const papi_attribute_t** attrs,
2520             void** iterator,
2521             const char* name,
2522             const papi_attribute_value_type_t type,
2523             papi_attribute_value_t** value );
2524
```

2525

2526 **Inputs**

2527

2528 attrs

2529 The attribute list.

2530 iterator

2531 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
2532 then only the first value is returned, even if the attribute is multivalued. If the
2533 argument points to a void* that is set to NULL, then the first attribute value is
2534 returned and the iterator can then be passed in unchanged on subsequent calls
2535 to this function to get the remaining values.

2536 name
 2537 Points to the name of the attribute whose value to get.

2538 type
 2539 The type of values for this attribute.

2540

2541 **Outputs**

2542

2543 value
 2544 Points to the variable where a pointer to the attribute value is to be returned.
 2545 Note that the returned pointer points to the attribute's value in the list (no copy
 2546 of the value is made) so that the caller does not need to do any special cleanup
 2547 of the returned value's memory (it is cleaned up when the containing attribute
 2548 list is freed).
 2549 If this call returns an error, the output value is not changed.

2550

2551 **Returns**

2552 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2553 value is returned.

2554 **Example**

2555

```

2556 #include "papi.h"
2557
2558 papi_attribute_t** attrs = NULL;
2559 papi_attribute_value_t* job_name_value;
2560 ...
2561 papiAttributeListGetValue(attrs,
2562     NULL,
2563     "job-name",
2564     PAPI_STRING,
2565     &job_name_value );
2566 if (job_name_value != NULL)
2567 {
2568     /* process the value */
2569     ...
2570 }
2571 ...
2572 papiAttributeListFree(attrs);
2573
  
```

2574

2575 **See Also**

2576 papiAttributeListFree, papiAttributeListGetString, papiAttributeListGetInteger,
 2577 papiAttributeListGetBoolean, papiAttributeListGetRange,
 2578 papiAttributeListGetResolution, papiAttributeListGetDatetime

2579 **6.11. papiAttributeListGetString**

2580 **Description**

2581 Get a string-valued attribute's value from an attribute list.

2582 **Syntax**

2583

```

2584     papi_status_t papiAttributeListGetString(
2585         const papi_attribute_t** attrs,
2586         void** iterator,
2587         const char* name,
2588         char** value );
2589

```

2590

2591 **Inputs**

2592

2593 attrs

2594 The attribute list.

2595 iterator

2596 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2597 then only the first value is returned, even if the attribute is multivalued. If the
 2598 argument points to a void* that is set to NULL, then the first attribute value is
 2599 returned and the iterator can then be passed in unchanged on subsequent calls
 2600 to this function to get the remaining values.

2601 name

2602 Points to the name of the attribute whose value to get.

2603

2604 **Outputs**

2605

2606 value

2607 Pointer to the char* where a pointer to the value is returned. If this call returns
 2608 an error, the output value is not changed.

2609

2610 **Returns**

2611 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2612 value is returned.

2613 **Example**

2614

```

2615 #include "papi.h"
2616
2617 papi_attribute_t** attrs = NULL;
2618 char* job_name_value = NULL;
2619 ...
2620 papiAttributeListGetString(attrs,
2621     NULL,
2622     "job-name",
2623     &job_name_value );
2624 if (job_name_value != NULL)
2625 {
2626     /* process the value */
2627     ...
2628 }
2629 ...
2630 papiAttributeListFree(attrs);
2631

```

2632

2633

See Also

2634

papiAttributeListFree, papiAttributeListGetValue

2635

6.12. papiAttributeListGetInteger

2636

Description

2637

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

2638

Syntax

2639

2640

```
papi_status_t papiAttributeListGetInteger(
2641     const papi_attribute_t** attrs,
2642     void** iterator,
2643     const char* name,
2644     int* value );
2645
```

2646

2647

Inputs

2648

2649

attrs

2650

The attribute list.

2651

iterator

2652

(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.

2653

2654

2655

2656

2657

name

2658

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

2659

2660

Outputs

2661

2662

value

2663

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

2664

2665

2666

Returns

2667

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

2668

2669

Example

2670

2671

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

2672

```

2673     papi_attribute_t** attrs = NULL;
2674     int copies = 0;
2675     ...
2676     papiAttributeListGetInteger(attrs,
2677                               NULL,
2678                               "copies",
2679                               &copies );
2680     /* process the value */
2681     ...
2682     papiAttributeListFree(attrs);
2683

```

2684

2685 **See Also**

2686 papiAttributeListFree, papiAttributeListGetValue

2687 **6.13. papiAttributeListGetBoolean**2688 **Description**

2689 Get an boolean-valued attribute's value from an attribute list.

2690 **Syntax**

2691

```

2692     papi_status_t papiAttributeListGetBoolean(
2693         const papi_attribute_t** attrs,
2694         void** iterator,
2695         const char* name,
2696         char* value );
2697

```

2698

2699 **Inputs**

2700

2701 attrs

2702 The attribute list.

2703 iterator

2704 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2705 then only the first value is returned, even if the attribute is multivalued. If the
 2706 argument points to a void* that is set to NULL, then the first attribute value is
 2707 returned and the iterator can then be passed in unchanged on subsequent calls
 2708 to this function to get the remaining values.

2709 name

2710 Points to the name of the attribute whose value to get.

2711

2712 **Outputs**

2713

2714 value

2715 Pointer to the char where the value is returned. If this call returns an error, the
 2716 output value is not changed.

2717

2718

Returns

2719

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

2720

2721

Example

2722

2723

```
#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);
```

2724

2725

2726

2727

2728

2729

2730

2731

2732

2733

2734

2735

2736

2737

See Also

2738

papiAttributeListFree, papiAttributeListGetValue

2739

6.14. papiAttributeListGetRange

2740

Description

2741

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

2742

Syntax

2743

2744

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

2745

2746

2747

2748

2749

2750

2751

2752

Inputs

2753

2754

attrs

The attribute list.

2755

2756

iterator

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

2757

2758

2759

2760

2761

2762

name

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

2763

2764

2765

Outputs

2766

2767 lower

2768

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

2769

2770 upper

2771

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

2772

2773

2774

Returns

2775

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

2776

2777

Example

2778

2779

```
#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);
```

2780

2781

2782

2783

2784

2785

2786

2787

2788

2789

2790

2791

2792

2793

2794

2795

See Also

2796

papiAttributeListFree, papiAttributeListGetValue

2797

6.15. papiAttributeListGetResolution

2798

Description

2799

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

2800

Syntax

2801

2802

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

2803

2804

2805

2806

2807

2808

2809

2810

2811 **Inputs**

2812

2813 attrs

2814 The attribute list.

2815 iterator

2816 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2817 then only the first value is returned, even if the attribute is multivalued. If the
 2818 argument points to a void* that is set to NULL, then the first attribute value is
 2819 returned and the iterator can then be passed in unchanged on subsequent calls
 2820 to this function to get the remaining values.

2821 name

2822 Points to the name of the attribute whose value to get.

2823

2824 **Outputs**

2825

2826 xres

2827 Pointer to the int where the X-resolution value is returned. If this call returns
 2828 an error, the output value is not changed.

2829 yres

2830 Pointer to the int where the Y-resolution value is returned. If this call returns
 2831 an error, the output value is not changed.

2832 units

2833 Pointer to the variable where the resolution-units value is returned. If this call
 2834 returns an error, the output value is not changed.

2835

2836 **Returns**

2837 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2838 value is returned.

2839 **Example**

2840

```

2841 #include "papi.h"
2842
2843 papi_attribute_t** attrs = NULL;
2844 int xres = 0;
2845 int yres = 0;
2846 papi_res_t units;
2847 ...
2848 papiAttributeListGetResolution(attrs,
2849     NULL,
2850     "printer-resolution",
2851     &xres,
2852     &yres,
2853     &units );
2854 /* process the value */
2855 ...
2856 papiAttributeListFree(attrs);
2857

```

2858

2859

See Also

2860

papiAttributeListFree, papiAttributeListGetValue

2861

6.16. papiAttributeListGetDatetime

2862

Description

2863

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

2864

Syntax

2865

2866

```
papi_status_t papiAttributeListGetDatetime(  
2867     const papi_attribute_t** attrs,  
2868     void** iterator,  
2869     const char* name,  
2870     time_t* date_time );  
2871
```

2872

2873

Inputs

2874

2875

attrs

2876

The attribute list.

2877

iterator

2878

(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.

2879

2880

2881

2882

2883

name

2884

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

2885

2886

Outputs

2887

2888

date_time

2889

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

2890

2891

2892

Returns

2893

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

2894

2895

Example

2896


```

2897     #include "papi.h"
2898
2899     papi_attribute_t** attrs = NULL;
2900     time_t date_time;
2901     ...
2902     papiAttributeListGetDatetime(attrs,
2903                                 NULL,
2904                                 "date-time-at-creation",
2905                                 &date_time );
2906     /* process the value */
2907     ...
2908     papiAttributeListFree(attrs);
2909

```

2910

2911 **See Also**

2912 papiAttributeListFree, papiAttributeListGetValue

2913 **6.17. papiAttributeListGetCollection**2914 **Description**

2915 Get a collection-valued attribute's value from an attribute list.

2916 **Syntax**

2917

```

2918     papi_status_t papiAttributeListGetCollection(
2919         const papi_attribute_t** attrs,
2920         void** iterator,
2921         const char* name,
2922         papi_attribute_t*** collection );
2923

```

2924

2925 **Inputs**

2926

2927 attrs

2928 The attribute list.

2929 iterator

2930 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2931 then only the first value is returned, even if the attribute is multivalued. If the
 2932 argument points to a void* that is set to NULL, then the first attribute value is
 2933 returned and the iterator can then be passed in unchanged on subsequent calls
 2934 to this function to get the remaining values.

2935 name

2936 Points to the name of the attribute whose value to get.

2937

2938 **Outputs**

2939

2940 collection

2941 Pointer to the attribute list where a pointer to the collection value is returned.
 2942 Note that the value is not copied, so the caller does not need to free the
 2943 returned list (it will be freed when the containing attribute list is freed).

2944 If this call returns an error, the output value is not changed.

2945

2946 **Returns**

2947 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2948 value is returned.

2949 **Example**

2950

```

2951 #include "papi.h"
2952
2953 papi_attribute_t** attrs = NULL;
2954 papi_attribute_t** collection = NULL;
2955 ...
2956 papiAttributeListGetCollection(attrs,
2957                               NULL,
2958                               "media-col",
2959                               &collection );
2960 /* process the value */
2961 ...
2962 papiAttributeListFree(attrs);
2963

```

2964

2965 **See Also**

2966 papiAttributeListFree, papiAttributeListGetValue

2967 **6.18. papiAttributeListFree**

2968 **Description**

2969 Frees an attribute list.

2970 **Syntax**

2971

```

2972 void papiAttributeListFree(
2973     const papi_attribute_t** attrs );
2974

```

2975

2976 **Inputs**

2977

2978 attrs

2979 Attribute list to be freed.

2980

2981 **Outputs**

2982 none

2983

Returns

2984

none

2985

Example

2986

2987

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

2988

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

2989

```
...
```

2990

```
papiAttributeListAddString(&attrs,
```

2991

```
    "job-name",
```

2992

```
    PAPI_EXCL,
```

2993

```
    1,
```

2994

```
    "My job" );
```

2995

```
...
```

2996

```
papiAttributeListFree(attrs);
```

2997

2998

2999

See Also

3000

papiAttributeListAddString, etc.

3001

6.19. papiAttributeListFind

3002

Description

3003

Find an attribute in an attribute list.

3004

Syntax

3005

3006

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

3007

3008

3009

3010

3011

Inputs

3012

3013

3014

attrs

Attribute list to be searched.

3015

3016

name

Pointer to the name of the attribute to find.

3017

3018

Outputs

3019

none

3020

Returns

3021

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

3022

3023

3024

Example

3025

```

3026     #include "papi.h"
3027
3028     papi_attribute_t** attrs = NULL;
3029     papi_attribute_t* attr = NULL;
3030     ...
3031     attr = papiAttributeListFind(&attrs,
3032                                 "job-name" );
3033
3034     if (attr != NULL)
3035     {
3036         /* process the attribute */
3037         ...
3038     }
3039     ...
3040     papiAttributeListFree(attrs);

```

3041

See Also

3042

papiAttributeListGetNext

3043

6.20. papiAttributeListGetNext

3044

Description

3045

Get the next attribute in an attribute list.

3046

Syntax

3047

3048

```

3049     papi_attribute_t* papiAttributeListGetNext(
3050         const papi_attribute_t** attrs,
3051         void** iterator );
3052

```

3053

Inputs

3054

3055

3056 attrs

Attribute list to be used.

3057

3058 iterator

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

3059

3060

3061

Outputs

3062

none

3063

Returns

3064

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

3065

3066

Example

3067

3068

```

3069     #include "papi.h"
3070
3071     papi_attribute_t** attrs = NULL;
3072     papi_attribute_t* attr = NULL;
3073     void* iterator = NULL;

```

```

3074     ...
3075     attr = papiAttributeListGetNext (&attrs,
3076                                     &iterator );
3077     while (attr != NULL)
3078     {
3079         /* process this attribute */
3080         ...
3081         attr = papiAttributeListGetNext (&attrs,
3082                                         &iterator );
3083     }
3084     ...
3085     papiAttributeListFree (attrs);
3086

```

3087

3088 **See Also**

3089 papiAttributeListFind

3090 **6.21. papiAttributeListFromString**3091 **Description**

3092 Convert a string of text options to an attribute list.

3093 PAPI provides two functions which map job attributes to and from text options
3094 that are typically provided on the command-line by the user. This text encoding is
3095 also backwards-compatible with existing printing systems and is relatively simple
3096 to parse and generate. See Appendix A for a definition of the string syntax.

3097 **Syntax**

3098

```

3099     papi_status_t papiAttributeListFromString(
3100                 papi_attribute_t*** attrs,
3101                 const int add_flags,
3102                 const char* buffer );
3103

```

3104

3105 **Inputs**

3106

3107 attrs

3108 Points to an attribute list. attrs equal to NULL is a bad argument, but if *attrs is
3109 NULL then this function will allocate the attribute list.

3110 add_flags

3111 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
3112 that indicates how to handle the request.

3113 buffer

3114 Points to text options.

3115

3116 **Outputs**

3117

3118 attrs

3119 The attribute list is updated.

3120

3121 **Returns**

3122 If the text string is successfully converted to an attribute list, a value of PAPI_OK is
3123 returned. Otherwise an appropriate failure value is returned.

3124 **Example**

3125

```
3126 #include "papi.h"
3127
3128 papi_attribute_t** attrs = NULL;
3129 char buffer[8192];
3130 ...
3131 strcpy(buffer,
3132 "copies=1 job-name=John's\ Really\040Nice\ Job");
3133
3134 papiAttributeListFromString(&attrs, PAPI_EXCL, buffer);
3135 ...
3136 papiAttributeListFree(attrs);
3137
```

3138

3139 **See Also**

3140 papiAttributeListToString

3141 6.22. papiAttributeListToString

3142 **Description**

3143 Convert an attribute list to its text representation. The destination string is limited
3144 to at most (buflen - 1) bytes plus the trailing nul byte.

3145 PAPI provides two functions which map job attributes to and from text options
3146 that are typically provided on the command-line by the user. This text encoding is
3147 also backwards-compatible with existing printing systems and is relatively simple
3148 to parse and generate. See Appendix A for a definition of the string syntax.

3149 **Syntax**

3150

```
3151 papi_status_t papiAttributeListToString(
3152     const papi_attribute_t** attrs,
3153     const char* attr_delim,
3154     char* buffer,
3155     const size_t buflen );
3156
```

3157

3158 **Inputs**

3159

3160 attrs

3161 Points to an attribute list.

3162 attr_delim
 3163 (optional) If not NULL, points to a string to be placed between attributes in the
 3164 output buffer. If NULL, a space is used as the attribute delimiter.

3165 buffer
 3166 Points to a string buffer to receive the to receive the text representation of the
 3167 attribute list.

3168 buflen
 3169 Specifies the length of the string buffer in bytes.
 3170

3171 **Outputs**

3172
 3173 buffer
 3174 The buffer is filled with the text representation of the attribute list. The buffer
 3175 will always be set to something by this function (buffer[0] = NULL in cases of
 3176 an error).
 3177

3178 **Returns**

3179 If the attribute list is successfully converted to a text string, a value of PAPI_OK is
 3180 returned. Otherwise an appropriate failure value is returned.

3181 **Example**

3182
 3183 #include "papi.h"
 3184
 3185 papi_attribute_t** attrs = NULL;
 3186 char buffer[8192];
 3187 ...
 3188 papiAttributeListToString(attrs, NULL, buffer, sizeof(buffer));
 3189 ...
 3190 papiAttributeListFree(attrs);
 3191

3192

3193 **See Also**

3194 papiAttributeListFromString

3195 Chapter 7. Job API

3196 7.1. papiJobSubmit

3197 Description

3198 Submits a print job having the specified attributes to the specified printer. This
3199 interface copies the specified print files before returning to the caller (contrast to
3200 papiJobSubmitByReference). The caller must call papiJobFree when done in order to
3201 free the resources associated with the returned job object.

3202 Attributes of the print job may be passed in the job_attributes argument and/or in
3203 a job ticket (using the job_ticket argument). If both are specified, the attributes in the
3204 job_attributes list will be applied to the job_ticket attributes and the resulting
3205 attribute set will be used.

3206 Semantics Reference

3207 Print-Job in [RFC2911], section 3.2.1

3208 Syntax

3209

```
3210 papi_status_t papiJobSubmit(  
3211     papi_service_t handle,  
3212     const char* printer_name,  
3213     const papi_attribute_t** job_attributes,  
3214     const papi_job_ticket_t* job_ticket,  
3215     const char** file_names,  
3216     papi_job_t* job );  
3217
```

3218

3219 Inputs

3220

3221 handle

3222 Handle to the print service to use.

3223 printer_name

3224 Pointer to the name of the printer to which the job is to be submitted.

3225 job_attributes

3226 (optional) The list of attributes describing the job and how it is to be printed. If
3227 options are specified here and also in the job ticket data, the value specified
3228 here takes precedence. If this is NULL then only default attributes and
3229 (optionally) a job ticket is submitted with the job.

3230 job_ticket

3231 (optional) Pointer to structure specifying the job ticket. If this argument is
3232 NULL, then no job ticket is used with the job.

3233 Whether the implementation passes both the attributes and the job ticket to the
3234 server/printer, or merges them to some print protocol or internal
3235 implementation depends on the implementation.

3236 file_names

3237 NULL terminated list of pointers to names of files to print. If more than one
 3238 file is specified, the files will be treated by the print system as separate
 3239 "documents" for things like page breaks and separator sheets, but they will be
 3240 scheduled and printed together as one job and the specified attributes will
 3241 apply to all the files.

3242 These file names may contain absolute path names or relative path names
 3243 (relative to the current path). The implementation **MUST** copy the file contents
 3244 before returning.

3245

3246 **Outputs**

3247

3248 job

3249 The resulting job object representing the submitted job.

3250

3251 **Returns**

3252 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3253 value is returned.

3254 **Example**

3255

```

3256 #include "papi.h"
3257
3258 papi_status_t status;
3259 papi_service_t handle = NULL;
3260 const char* printer = "my-printer";
3261 const papi_attribute_t** attrs = NULL;
3262 const papi_job_ticket_t* ticket = NULL;
3263 const char* files[] = { "/etc/motd", NULL };
3264 papi_job_t job = NULL;
3265
3266 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3267                           PAPI_ENCRYPT_IF_REQUESTED, NULL);
3268 if (status != PAPI_OK)
3269 {
3270     /* handle the error */
3271     ...
3272 }
3273
3274 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3275                             PAPI_STRING, 1, "test job");
3276 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3277                              PAPI_INTEGER, 1, 4);
3278
3279 status = papiJobSubmit(handle,
3280                        printer,
3281                        attrs,
3282                        ticket,
3283                        files,
3284                        &job);
3285 if (status != PAPI_OK)
3286 {
3287     fprintf(stderr, "papiJobSubmit failed: %s\n",
3288            papiStatusString(status));
3289     ...
3290 }
3291
3292 if (job != NULL)
3293 {
3294     /* look at the job object (maybe get the id) */
3295     papiJobFree(job);
3296 }
3297
3298 papiServiceDestroy(handle);
3299
3300
```

3301

3302

See Also

3303

papiJobSubmitByReference, papiJobValidate, papiJobFree

3304

7.2. papiJobSubmitByReference

3305

Description

3306

3307

3308

3309

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).

3310

3311

3312

3313

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.

3314

Semantics Reference

3315

Print-URI in [RFC2911], section 3.2.2

3316

Syntax

3317

3318

3319

3320

3321

3322

3323

3324

3325

```
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 );
```

3326

3327

Inputs

3328

3329

handle

3330

Handle to the print service to use.

3331

printer_name

3332

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

3333

job_attributes

3334

3335

3336

3337

(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.

3338

job_ticket

3339

3340

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

3341 Whether the implementation passes both the attributes and the job ticket to the
 3342 server/printer, or merges them to some print protocol or internal
 3343 implementation depends on the implementation.

3344 file_names

3345 NULL terminated list of pointers to names of files to print. If more than one
 3346 file is specified, the files will be treated by the print system as separate
 3347 "documents" for things like page breaks and separator sheets, but they will be
 3348 scheduled and printed together as one job and the specified attributes will
 3349 apply to all the files.

3350 These file names may contain absolute path names, relative path names or
 3351 URIs ([RFC1738], [RFC2396]). The implementation SHOULD NOT copy the
 3352 referenced data unless (or until) it is no longer feasible to maintain the
 3353 reference. Feasibility limitations may arise out of security issues, namespace
 3354 issues, and/or protocol or printer limitations.

3355 Implementations MUST support the absolute path, relative path, and "file:"
 3356 URI scheme. Use of other URI schemes could result in a PAPI_URI_SCHEME
 3357 error, depending on the implementation.

3358 The semantics explained in the preceding paragraphs allows for flexibility in
 3359 the PAPI implementation. For example: (1) PAPI on top of a local service to
 3360 maintain the reference for the life of the job, if the local service supports it. (2)
 3361 PAPI on top of IPP to send a reference when the server can access the
 3362 referenced data and copy it when it is not accessible to the server. (3) PAPI on
 3363 top of network printing protocols that don't support references to copy the data
 3364 on the way out to the remote server.

3365

3366 **Outputs**

3367

3368 job

3369 The resulting job object representing the submitted job.

3370

3371 **Returns**

3372 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3373 value is returned.

3374 **Example**

3375

```

3376 #include "papi.h"
3377
3378 papi_status_t status;
3379 papi_service_t handle = NULL;
3380 const char* printer = "my-printer";
3381 const papi_attribute_t** attrs = NULL;
3382 const papi_job_ticket_t* ticket = NULL;
3383 const char* files[] = { "http://foo.bar.org/docs/glop.pdf", NULL };
3384 papi_job_t job = NULL;
3385
3386 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3387                           PAPI_ENCRYPT_IF_REQUESTED, NULL);
3388 if (status != PAPI_OK)
3389 {
3390     /* handle the error */
3391     ...
3392 }

```

```

3393
3394     papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3395                               PAPI_STRING, 1, "test job");
3396     papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3397                                PAPI_INTEGER, 1, 4);
3398
3399     status = papiJobSubmitByReference(handle,
3400                                     printer,
3401                                     attrs,
3402                                     ticket,
3403                                     files,
3404                                     &job);
3405
3406     if (status != PAPI_OK)
3407     {
3408         fprintf(stderr, "papiJobSubmitByReference failed: %s\n",
3409                papiStatusString(status));
3410         ...
3411     }
3412
3413     if (job != NULL)
3414     {
3415         /* look at the job object (maybe get the id) */
3416         papiJobFree(job);
3417     }
3418
3419     papiServiceDestroy(handle);
3420

```

3421

3422

See Also

3423

papiJobSubmit, papiJobValidate, papiJobFree

3424

7.3. papiJobValidate

3425

Description

3426

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.

3427

3428

3429

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.

3430

3431

3432

3433

Semantics Reference

3434

Validate-Job in [RFC2911], section 3.2.3

3435

Syntax

3436

3437

```

3438     papi_status_t papiJobValidate(
3439                 papi_service_t      handle,
3440                 const char*          printer_name,
3441                 const papi_attribute_t** job_attributes,
3442                 const papi_job_ticket_t* job_ticket,
3443                 const char**         file_names,
3444                 papi_job_t*          job );

```

3445

3446

Inputs

3447

3448 handle
 3449 Handle to the print service to use.

3450 printer_name
 3451 Pointer to the name of the printer against which the job is to be validated.

3452 job_attributes
 3453 (optional) The list of attributes describing the job and how it is to be printed. If
 3454 options are specified here and also in the job ticket data, the value specified
 3455 here takes precedence. If this is NULL then only default attributes and
 3456 (optionally) a job ticket is submitted with the job.

3457 job_ticket
 3458 (optional) Pointer to structure specifying the JDF job ticket. If this argument is
 3459 NULL, then no job ticket is used with the job.

3460 file_names
 3461 NULL terminated list of pointers to names of files to validate.

3463 Outputs

3464

3465 job
 3466 The resulting job object representing what would be the submitted job.

3468 Returns

3469 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3470 value is returned.

3471 Example

```

3472
3473 #include "papi.h"
3474
3475 papi_status_t status;
3476 papi_service_t handle = NULL;
3477 const char* printer = "my-printer";
3478 const papi_attribute_t** attrs = NULL;
3479 const papi_job_ticket_t* ticket = NULL;
3480 const char* files[] = { "/etc/motd", NULL };
3481 papi_job_t job = NULL;
3482
3483 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3484                           PAPI_ENCRYPT_IF_REQUESTED, NULL);
3485 if (status != PAPI_OK)
3486 {
3487     /* handle the error */
3488     ...
3489 }
3490
3491 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3492                            PAPI_STRING, 1, "test job");
3493 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3494                             PAPI_INTEGER, 1, 4);
3495
3496 status = papiJobValidate(handle,
3497                          printer,
3498                          attrs,
3499                          ticket,
3500                          files,
3501                          &job);
  
```

```

3502     if (status != PAPI_OK)
3503     {
3504         fprintf(stderr, "papiJobValidate failed: %s\n",
3505                papiStatusString(status));
3506         ...
3507     }
3508
3509     if (job != NULL)
3510     {
3511         ...
3512         papiJobFree(job);
3513     }
3514
3515     papiServiceDestroy(handle);
3516

```

3517

3518

See Also

3519

papiJobSubmit, papiJobFree

3520

7.4. papiJobStreamOpen

3521

Description

3522

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.

3523

3524

3525

3526

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

3527

3528

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.

3529

3530

3531

3532

Syntax

3533

3534

```

papi_status_t papiJobStreamOpen(
3535     papi_service_t      handle,
3536     const char*         printer_name,
3537     const papi_attribute_t** job_attributes,
3538     const papi_job_ticket_t* job_ticket,
3539     papi_stream_t*     stream );
3540

```

3541

3542

Inputs

3543

3544

handle

3545

Handle to the print service to use.

3546

printer_name

3547

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

3548 job_attributes
 3549 (optional) The list of attributes describing the job and how it is to be printed.
 3550 See job_attributes argument for papiJobSubmit for description.

3551 job_ticket
 3552 (optional) Pointer to structure specifying the job ticket. See job_ticket argument
 3553 for papiJobSubmit for description.

3554

3555 Outputs

3556

3557 stream

3558 The resulting stream object to which print data can be written.

3559

3560 Returns

3561 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3562 value is returned.

3563 Example

3564

```

3565 #include "papi.h"
3566
3567 papi_status_t status;
3568 papi_service_t handle = NULL;
3569 const char* printer = "my-printer";
3570 const papi_attribute_t** attrs = NULL;
3571 const papi_job_ticket_t* ticket = NULL;
3572 papi_stream_t stream = NULL;
3573 papi_job_t job = NULL;
3574 char buffer[4096];
3575 size_t buflen = 0;
3576
3577 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3578                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
3579 if (status != PAPI_OK)
3580 {
3581     /* handle the error */
3582     ...
3583 }
3584
3585 papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3586                          PAPI_STRING, 1, "test job");
3587 papiAttributeListAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
3588                          PAPI_INTEGER, 1, 4);
3589
3590 /* Open the print job stream */
3591 status = papiJobStreamOpen(handle,
3592                          printer,
3593                          attrs,
3594                          ticket,
3595                          &stream);
3596
3597 if (status != PAPI_OK)
3598 {
3599     fprintf(stderr, "papiJobStreamOpen failed: %s\n",
3600           papiStatusString(status));
3601     ...
3602 }
3603
3604 /* Write all the print job data */
3605 while(print_data_remaining)
3606 {
3607     /* Generate the print data */
3608     ...
3609     /* Write the print data */
3610     status = papiJobStreamWrite(handle
3611                               stream,
3612                               buffer,
3613                               buflen);

```

```

3613         if (status != PAPI_OK)
3614         {
3615             fprintf(stderr, "papiJobStreamWrite failed: %s\n",
3616                     papiStatusString(status));
3617             ...
3618         }
3619     }
3620
3621     /* Close the print job stream */
3622     status = papiJobStreamClose(handle, stream, &job);
3623     if (status != PAPI_OK)
3624     {
3625         fprintf(stderr, "papiJobStreamClose failed: %s\n",
3626                 papiStatusString(status));
3627         ...
3628     }
3629
3630     papiJobFree(job);
3631     papiServiceDestroy(handle);
3632

```

3633

3634 **See Also**

3635 papiJobStreamWrite, papiJobStreamClose

3636 **7.5. papiJobStreamWrite**3637 **Description**

3638 Writes print data to the specified open job stream. The open job stream must have
 3639 been obtained by a successful call to papiJobStreamOpen.

3640 **Syntax**

3641

```

3642 papi_status_t papiJobStreamWrite(
3643             papi_service_t   handle,
3644             papi_stream_t    stream,
3645             const void*      buffer,
3646             const size_t     buflen );
3647

```

3648

3649 **Inputs**

3650

3651 handle

3652 Handle to the print service to use.

3653 stream

3654 The open stream object to which print data is written.

3655 buffer

3656 Pointer to the buffer of print data to write.

3657 buflen

3658 The number of bytes to write.

3659

3660 **Outputs**

3661 none

3662 **Returns**
 3663 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3664 value is returned.

3665 **Example**

3666 See papiJobStreamOpen

3667 **See Also**

3668 papiJobStreamOpen, papiJobStreamClose

3669 **7.6. papiJobStreamClose**

3670 **Description**

3671 Closes the specified open job stream and completes submission of the job (if there
 3672 were no previous errors returned from papiJobSubmitWrite). The open job stream
 3673 must have been obtained by a successful call to papiJobStreamOpen.

3674 **Syntax**

3675

```

3676           papi_status_t papiJobStreamClose(
3677                           papi_service_t        handle,
3678                           papi_stream_t         stream,
3679                           papi_job_t*          job );
3680
```

3681

3682 **Inputs**

3683

3684 handle

3685 Handle to the print service to use.

3686 stream

3687 The open stream object to which print data was written.

3688

3689 **Outputs**

3690

3691 job

3692 The resulting job object representing the submitted job.

3693

3694 **Returns**

3695 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3696 value is returned.

3697 **Example**

3698 See papiJobStreamOpen

3699 **See Also**
3700 papiJobStreamOpen, papiJobStreamWrite

3701 **7.7. papiJobQuery**

3702 **Description**

3703 Queries some or all the attributes of the specified job object.

3704 **Semantics Reference**

3705 Get-Job-Attributes in [RFC2911], section 3.3.4

3706 **Syntax**

3707

```
3708           papi_status_t papiJobQuery(  
3709                           papi_service_t     handle,  
3710                           const char*       printer_name,  
3711                           const int32_t     job_id,  
3712                           const char*       requested_attrs[],  
3713                           papi_job_t*       job );  
3714
```

3715

3716 **Inputs**

3717

3718 handle

3719 Handle to the print service to use.

3720 printer_name

3721 Pointer to the name or URI of the printer to which the job was submitted.

3722 job_id

3723 The ID number of the job to be queried.

3724 requested_attrs

3725 NULL terminated array of attributes to be queried. If NULL is passed then all
3726 available attributes are queried. (NOTE: The job may return more attributes
3727 than you requested. This is merely an advisory request that may reduce the
3728 amount of data returned if the printer/server supports it.)

3729

3730 **Outputs**

3731

3732 job

3733 The returned job object containing the requested attributes.

3734

3735 **Returns**

3736 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3737 value is returned.

3738

Example

3739

```

3740 #include "papi.h"
3741
3742 papi_status_t status;
3743 papi_service_t handle = NULL;
3744 const char* printer_name = "my-printer";
3745 papi_job_t job = NULL;
3746 int32_t job_id = 12;
3747 const char* job_attrs[] =
3748 {
3749     "job-id",
3750     "job-name",
3751     "job-originating-user-name",
3752     "job-state",
3753     "job-state-reasons",
3754     NULL
3755 };
3756 ...
3757 status = papiServiceCreate(&handle,
3758                             NULL,
3759                             NULL,
3760                             NULL,
3761                             NULL,
3762                             PAPI_ENCRYPT_NEVER,
3763                             NULL);
3764
3765 if (status != PAPI_OK)
3766 {
3767     /* handle the error */
3768     ...
3769 }
3770
3771 status = papiJobQuery(handle,
3772                       printer_name,
3773                       job_id,
3774                       job_attrs,
3775                       &job);
3776
3777 if (status != PAPI_OK)
3778 {
3779     /* handle the error */
3780     fprintf(stderr, "papiJobQuery failed: %s\n",
3781            papiServiceGetStatusMessage(handle));
3782     ...
3783 }
3784
3785 if (job != NULL)
3786 {
3787     /* process the job */
3788     ...
3789     papiJobFree(job);
3790 }
3791
3792 papiServiceDestroy(handle);

```

3792

3793

See Also

3794

papiJobFree, papiPrinterListJobs, papiJobModify

3795

7.8. papiJobModify

3796

Description

3797

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.

3798

3799

3800

Semantics Reference

3801

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

3802

Syntax

3803

3804

papi_status_t papiJobModify(

```

3805         papi_service_t    handle,
3806         const char*        printer_name,
3807         const int32_t       job_id,
3808         const papi_attribute_t** attrs,
3809         papi_job_t*        job );
3810

```

3811

3812 **Inputs**

3813

3814 handle

3815 Handle to the print service to use.

3816 printer_name

3817 Pointer to the name or URI of the printer to which the job was submitted.

3818 job_id

3819 The ID number of the job to be modified.

3820 attrs

3821 Attributes to be modified. Any attributes not specified are left unchanged.

3822

3823 **Outputs**

3824

3825 job

3826 The modified job object.

3827

3828 **Returns**

3829 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3830 value is returned.

3831 **Example**

3832

```

3833 #include "papi.h"
3834
3835 papi_status_t status;
3836 papi_service_t handle = NULL;
3837 const char* printer_name = "my-printer";
3838 papi_job_t job = NULL;
3839 int32_t job_id = 12;
3840 papi_attribute_t** attrs = NULL;
3841 ...
3842 status = papiServiceCreate(&handle,
3843                          NULL,
3844                          NULL,
3845                          NULL,
3846                          NULL,
3847                          PAPI_ENCRYPT_NEVER,
3848                          NULL);
3849
3850 if (status != PAPI_OK)
3851 {
3852     /* handle the error */
3853     ...
3854 }
3855 papiAttributeListAddInteger(&attrs,

```

```

3856         PAPI_EXCL,
3857         "copies",
3858         3);
3859
3860     status = papiJobModify(handle,
3861                           printer_name,
3862                           job_id,
3863                           attrs,
3864                           &job);
3865     if (status != PAPI_OK)
3866     {
3867         /* handle the error */
3868         fprintf(stderr, "papiJobModify failed: %s\n",
3869                papiServiceGetStatusMessage(handle));
3870         ...
3871     }
3872
3873     if (job != NULL)
3874     {
3875         /* process the job */
3876         ...
3877         papiJobFree(job);
3878     }
3879
3880     papiServiceDestroy(handle);
3881

```

3882

3883 **See Also**

3884 papiJobQuery, papiJobFree, papiPrinterListJobs

3885 **7.9. papiJobCancel**3886 **Description**

3887 Cancel the specified print job.

3888 **Semantics Reference**

3889 Cancel-Job in [RFC2911], section 3.3.3

3890 **Syntax**

3891

```

3892 papi_status_t papiJobCancel(
3893             papi_service_t   handle,
3894             const char*      printer_name,
3895             const int32_t     job_id );
3896

```

3897

3898 **Inputs**

3899

3900 handle

3901 Handle to the print service to use.

3902 printer_name

3903 Pointer to the name or URI of the printer to which the job was submitted.

3904 job_id

3905 The ID number of the job to be cancelled.

3906

3907

Outputs

3908

none

3909

Returns

3910

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

3911

3912

Example

3913

3914

```

#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
int32_t job_id = 12;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);

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

status = papiJobCancel(handle,
                      printer_name,
                      job_id);

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

papiServiceDestroy(handle);

```

3915

3916

3917

3918

3919

3920

3921

3922

3923

3924

3925

3926

3927

3928

3929

3930

3931

3932

3933

3934

3935

3936

3937

3938

3939

3940

3941

3942

3943

3944

3945

3946

3947

3948

See Also

3949

papiPrinterListJobs, papiPrinterPurgeJobs

3950

7.10. papiJobHold

3951

Description

3952

Holds the specified print job and prevents it from being scheduled for printing.

3953

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

3954

papiJobRelease to undo the effects of this operation, or specify the hold_until

3955

argument to automatically release the job at a specific time.

3956

Semantics Reference

3957

Hold-Job in [RFC2911], section 3.3.5

3958

Syntax

3959

3960

```
papi_status_t papiJobHold(
```

3961

```
    papi_service_t    handle,
```

3962

```
    const char*      printer_name,
```

3963

```
    const int32_t    job_id,
```

3964

```
    const char*      hold_until,
```

```

3965         const time_t*         hold_until_time );
3966

```

3967

3968 **Inputs**

3969

3970 handle

3971 Handle to the print service to use.

3972 printer_name

3973 Pointer to the name or URI of the printer to which the job was submitted.

3974 job_id

3975 The ID number of the job to be held.

3976 hold_until

3977 (optional) Specifies the time when the job will be automatically released for
3978 printing. If NULL, the job is held until explicitly released by calling
3979 papiJobRelease. If specified, the value must be one of the strings "indefinite"
3980 (same effect as passing NULL), "day-time", "evening", "night", "weekend",
3981 "second-shift", "third-shift", or "timed". For values other than "indefinite" and
3982 "timed", the printer/server must define exact times associated with these
3983 values and it may make these associations configurable. If "timed" is specified,
3984 then the hold_until_time argument is used.

3985 hold_until_time

3986 (optional) Specifies the time when the job will be automatically released for
3987 printing. This argument is ignored unless "timed" is passed as the hold_until
3988 argument.

3989

3990 **Outputs**

3991 none

3992 **Returns**

3993 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3994 value is returned.

3995 **Example**

3996

```

3997 #include "papi.h"
3998
3999 papi_status_t status;
4000 papi_service_t handle = NULL;
4001 const char* printer_name = "my-printer";
4002 int32_t job_id = 12;
4003 ...
4004 status = papiServiceCreate(&handle,
4005                             NULL,
4006                             NULL,
4007                             NULL,
4008                             NULL,
4009                             PAPI_ENCRYPT_NEVER,
4010                             NULL);
4011 if (status != PAPI_OK)
4012 {

```

```

4013         /* handle the error */
4014         ...
4015     }
4016
4017     status = papiJobHold(handle,
4018                       printer_name,
4019                       job_id,
4020                       NULL,
4021                       NULL);
4022     if (status != PAPI_OK)
4023     {
4024         /* handle the error */
4025         fprintf(stderr, "papiJobHold failed: %s\n",
4026               papiServiceGetStatusMessage(handle));
4027         ...
4028     }
4029
4030     papiServiceDestroy(handle);
4031

```

4032

4033 **See Also**

4034 papiJobRelease

4035 **7.11. papiJobRelease**4036 **Description**

4037 Releases the specified print job, allowing it to be scheduled for printing. This
 4038 operation is optional and may not be supported by all printers/servers, but it must
 4039 be supported if papiJobHold is supported.

4040 **Semantics Reference**

4041 Release-Job in [RFC2911], section 3.3.6

4042 **Syntax**

4043

```

4044 papi_status_t papiJobRelease(
4045             papi_service_t      handle,
4046             const char*         printer_name,
4047             const int32_t        job_id );
4048

```

4049

4050 **Inputs**

4051

4052 handle

4053 Handle to the print service to use.

4054 printer_name

4055 Pointer to the name or URI of the printer to which the job was submitted.

4056 job_id

4057 The ID number of the job to be released.

4058

4059 **Outputs**

4060 none

4061

Returns

4062

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

4063

4064

Example

4065

4066

```

#include "papi.h"

papi_status_t status;
papi_service_t handle = NULL;
const char* printer_name = "my-printer";
int32_t job_id = 12;
...
status = papiServiceCreate(&handle,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           PAPI_ENCRYPT_NEVER,
                           NULL);

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

status = papiJobRelease(handle,
                       printer_name,
                       job_id);

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

papiServiceDestroy(handle);

```

4067

4068

4069

4070

4071

4072

4073

4074

4075

4076

4077

4078

4079

4080

4081

4082

4083

4084

4085

4086

4087

4088

4089

4090

4091

4092

4093

4094

4095

4096

4097

4098

4099

4100

See Also

4101

papiJobHold

4102

7.12. papiJobRestart

4103

Description

4104

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.

4105

4106

4107

Semantics Reference

4108

Restart-Job in [RFC2911], section 3.3.7

4109

Syntax

4110

4111

```
papi_status_t papiJobRestart(
```

4112

```
    papi_service_t    handle,
```

4113

```
    const char*      printer_name,
```

4114

```
    const int32_t    job_id );
```

4115

4116

4117 **Inputs**

4118

4119 handle

4120 Handle to the print service to use.

4121 printer_name

4122 Pointer to the name or URI of the printer to which the job was submitted.

4123 job_id

4124 The ID number of the job to be restarted.

4125

4126 **Outputs**

4127 none

4128 **Returns**4129 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
4130 value is returned.4131 **Example**

4132

```

4133 #include "papi.h"
4134
4135 papi_status_t status;
4136 papi_service_t handle = NULL;
4137 const char* printer_name = "my-printer";
4138 int32_t job_id = 12;
4139 ...
4140 status = papiServiceCreate(&handle,
4141                            NULL,
4142                            NULL,
4143                            NULL,
4144                            NULL,
4145                            PAPI_ENCRYPT_NEVER,
4146                            NULL);
4147
4148 if (status != PAPI_OK)
4149 {
4150     /* handle the error */
4151     ...
4152 }
4153
4154 status = papiJobRestart(handle,
4155                        printer_name,
4156                        job_id);
4157
4158 if (status != PAPI_OK)
4159 {
4160     /* handle the error */
4161     fprintf(stderr, "papiJobRestart failed: %s\n",
4162            papiServiceGetStatusMessage(handle));
4163     ...
4164 }
4165 papiServiceDestroy(handle);

```

4166

4167 **See Also**

4168 papiPrinterListJobs

4169 **7.13. papiJobGetAttributeList**4170 **Description**

4171 Get the attribute list associated with a job object.

4172 This function retrieves an attribute list from a job object returned in a previous call.
 4173 Job objects are returned as a result of the operations performed by
 4174 `papiPrinterListJobs`, `papiJobQuery`, `papiJobModify`, `papiJobSubmit`,
 4175 `papiJobSubmitByReference`, and `papiJobClose`.

4176 **Syntax**

4177

```
4178 papi_attribute_t** papiJobGetAttributeList(  
4179     papi_job_t     job );  
4180
```

4181

4182 **Inputs**

4183

4184 `job`

Handle of the job object.

4186

4187 **Outputs**

4188 none

4189 **Returns**

4190 Pointer to the attribute list associated with the job object.

4191 **Example**

4192

```
4193 #include "papi.h"  
4194  
4195 papi_status_t status;  
4196 papi_service_t handle = NULL;  
4197 const char* printer_name = "my-printer";  
4198 papi_job_t job = NULL;  
4199 papi_attribute_list* attrs = NULL;  
4200 ...  
4201 status = papiServiceCreate(&handle,  
4202     NULL,  
4203     NULL,  
4204     NULL,  
4205     NULL,  
4206     PAPI_ENCRYPT_NEVER,  
4207     NULL);  
4208  
4209 if (status != PAPI_OK)  
4210 {  
4211     /* handle the error */  
4212     ...  
4213 }  
4214  
4215 status = papiJobQuery(handle,  
4216     printer_name,  
4217     67,  
4218     NULL,  
4219     &job);  
4220  
4221 if (status != PAPI_OK)  
4222 {  
4223     /* handle the error */  
4224     fprintf(stderr, "papiJobQuery failed: %s\n",  
4225         papiServiceGetStatusMessage(handle));  
4226     ...  
4227 }  
4228  
4229 if (job != NULL)  
4230 {  
4231     /* process the job object */  
4232     attrs = papiJobGetAttributeList(job);  
4233     ...  
4234     papiJobFree(job);  
4235 }
```

```
4233     }  
4234  
4235     papiServiceDestroy(handle);  
4236
```

4237

4238 **See Also**

4239 papiPrinterListJobs, papiJobQuery

4240 7.14. papiJobGetPrinterName

4241 **Description**

4242 Get the printer name associated with a job object.

4243 **Syntax**

4244

```
4245     char* papiJobGetPrinterName(  
4246         papi_job_t    job );  
4247
```

4248

4249 **Inputs**

4250

4251 job

4252 Handle of the job object.

4253

4254 **Outputs**

4255 none

4256 **Returns**

4257 Pointer to the printer name associated with the job object.

4258 **Example**

4259

```
4260     #include "papi.h"  
4261  
4262     char* printer_name = NULL;  
4263     papi_job_t job = NULL;  
4264     ...  
4265     if (job != NULL)  
4266     {  
4267         /* process the job object */  
4268         printer_name = papiJobGetPrinterName(job);  
4269         ...  
4270         papiJobFree(job);  
4271     }  
4272
```

4273

4274 **See Also**

4275 papiPrinterListJobs, papiJobQuery

4276 **7.15. papiJobGetId**4277 **Description**

4278 Get the job ID associated with a job object.

4279 **Syntax**

4280

```

4281 int32_t papiJobGetId(
4282         papi_job_t    job );
4283 
```

4284

4285 **Inputs**

4286

4287 job

4288 Handle of the job object.

4289

4290 **Outputs**

4291 none

4292 **Returns**

4293 The job ID associated with the job object.

4294 **Example**

4295

```

4296 #include "papi.h"
4297
4298 int32_t job_id;
4299 papi_job_t job = NULL;
4300 ...
4301 if (job != NULL)
4302 {
4303     /* process the job object */
4304     job_id = papiJobGetId(job);
4305     ...
4306     papiJobFree(job);
4307 }
4308 
```

4309

4310 **See Also**

4311 papiPrinterListJobs, papiJobQuery

4312 **7.16. papiJobGetJobTicket**4313 **Description**

4314 Get the job ticket associated with a job object.

4315 **Syntax**

4316

```

4317 papi_job_ticket_t* papiJobGetJobTicket(
4318         papi_job_t    job );
4319 
```

4320

4321 **Inputs**

4322

4323 job

4324 Handle of the job object.

4325

4326 **Outputs**

4327 none

4328 **Returns**

4329 Pointer to the job ticket associated with the job object.

4330 **Example**

4331

```
4332           #include "papi.h"  
4333  
4334           papi_job_ticket_t* job_ticket = NULL;  
4335           papi_job_t job = NULL;  
4336           ...  
4337           if (job != NULL)  
4338           {  
4339               /* process the job object */  
4340               job_ticket = papiJobGetJobTicket(job);  
4341               ...  
4342               papiJobFree(job);  
4343           }  
4344
```

4345

4346 **See Also**

4347 papiPrinterListJobs, papiJobQuery

4348 **7.17. papiJobFree**

4349 **Description**

4350 Free a job object.

4351 **Syntax**

4352

```
4353           void papiJobFree(  
4354                            papi_job_t        job );  
4355
```

4356

4357 **Inputs**

4358

4359 job

4360 Handle of the job object to free.

4361

4362 **Outputs**

4363 none

4364 **Returns**

4365 none

4366 **Example**

4367

```

4368 #include "papi.h"
4369
4370 papi_status_t status;
4371 papi_service_t handle = NULL;
4372 const char* printer_name = "my-printer";
4373 papi_job_t job = NULL;
4374 ...
4375 status = papiServiceCreate(&handle,
4376                             NULL,
4377                             NULL,
4378                             NULL,
4379                             NULL,
4380                             PAPI_ENCRYPT_NEVER,
4381                             NULL);
4382
4383 if (status != PAPI_OK)
4384 {
4385     /* handle the error */
4386     ...
4387 }
4388
4389 status = papiJobQuery(handle,
4390                       printer_name,
4391                       12,
4392                       &job);
4393 if (status != PAPI_OK)
4394 {
4395     /* handle the error */
4396     fprintf(stderr, "papiJobQuery failed: %s\n",
4397            papiServiceGetStatusMessage(handle));
4398     ...
4399 }
4400
4401 if (job != NULL)
4402 {
4403     /* process the job object */
4404     ...
4405     papiJobFree(job);
4406 }
4407
4408 papiServiceDestroy(handle);

```

4409

4410 **See Also**

4411 papiJobQuery

4412 **7.18. papiJobListFree**

4413 **Description**

4414 Free a list of job objects.

4415 **Syntax**

4416

```

4417 void papiJobListFree(
4418     papi_job_t*   jobs );
4419

```

4420

4421 **Inputs**

4422

4423 jobs

4424 Pointer to the job object list to free.

4425

4426 **Outputs**

4427 none

4428 **Returns**

4429 none

4430 **Example**

4431

```

4432           #include "papi.h"
4433
4434           papi_status_t status;
4435           papi_service_t handle = NULL;
4436           const char* printer_name = "my-printer";
4437           papi_job_t* jobs = NULL;
4438           ...
4439           status = papiServiceCreate(&handle,
4440                                       NULL,
4441                                       NULL,
4442                                       NULL,
4443                                       NULL,
4444                                       PAPI_ENCRYPT_NEVER,
4445                                       NULL);
4446
4447           if (status != PAPI_OK)
4448           {
4449               /* handle the error */
4450               ...
4451           }
4452
4453           status = papiPrinterListJobs(handle,
4454                                        printer_name,
4455                                        NULL,
4456                                        0, 0, 0,
4457                                        &jobs);
4458
4459           if (status != PAPI_OK)
4460           {
4461               /* handle the error */
4462               fprintf(stderr, "papiPrinterListJobs failed: %s\n",
4463                        papiServiceGetStatusMessage(handle));
4464               ...
4465           }
4466
4467           if (jobs != NULL)
4468           {
4469               /* process the job objects */
4470               ...
4471               papiJobListFree(jobs);
4472           }
4473           papiServiceDestroy(handle);

```

4474

4475 **See Also**

4476 papiPrinterListJobs

4477 **Chapter 8. Miscellaneous API**

4478 **8.1. papiStatusString**

4479 **Description**

4480 Get a status string for the specified papi_status_t. The status message returned
4481 from this function may be less detailed than the status message returned from
4482 papiServiceGetStatusMessage (if the print service supports returning more detailed
4483 error messages).

4484 The returned message will be localized in the language of the submitter of the
4485 requestor.

4486 **Syntax**

4487

```
4488 char* papiStatusString(  
4489     const papi_status_t status );  
4490
```

4491

4492 **Inputs**

4493

4494 status

4495 The status value to convert to a status string.

4496

4497 **Outputs**

4498 none

4499 **Returns**

4500 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
4501 value is returned.

4502 **Example**

4503

```
4504 #include "papi.h"  
4505 papi_status_t status;  
4506 ...  
4507 fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));  
4508  
4509
```

4510

4511 **See Also**

4512 papiServiceGetStatusMessage

4513 Chapter 9. Attributes

4514 For a summary of the IPP attributes which can be used with the PAPI interface, see:
4515 <ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf>

4516 9.1. Extension Attributes

4517 The following attributes are not currently defined by IPP, but may be used with
4518 this API.

4519 9.1.1. job-ticket-formats-supported

4520 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats
4521 that are supported by the printer. If this attribute is not present, it is assumed that
4522 the printer does not support any job ticket formats.

4523

4524 9.2. Required Job Attributes

4525 The following job attributes *must* be supported to comply with this API standard.
4526 These attributes may be supported by the underlying print server directly, or they
4527 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

4528

4529 9.3. Required Printer Attributes

4530 The following printer attributes *must* be supported to comply with this API
4531 standard. These attributes may be supported by the underlying print server
4532 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

4533 queued-job-count
 uri-authentication-supported
 uri-security-supported

4534 **9.4. IPP Attribute Type Mapping**

4535 The following table maps IPP to PAPI attribute value types:

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

4536

4537 Appendix A. Attribute List Text Representation

4538 A.1. ABNF Definition

4539 The following ABNF definition [RFC2234] describes the syntax of PAPI attributes
4540 encoded as text options:

```
4541 OPTION-STRING = [OPTION] *(1*WC OPTION) *WC
4542
4543 OPTION       = TRUEOPTION / FALSEOPTION / VALUEOPTION
4544
4545 TRUEOPTION  = NAME
4546
4547 FALSEOPTION = "no" NAME
4548
4549 VALUEOPTION = NAME "=" VALUE *( "," VALUE )
4550
4551 NAME        = 1*NAMECHAR
4552
4553 NAMECHAR    = DIGIT / ALPHA / "-" / "_" / "."
4554
4555 VALUE       = BOOLVALUE / COLVALUE / DATEVALUE / NUMBEVALUE / QUOTEDVALUE /
4556             RANGEVALUE / RESVALUE / STRINGVALUE
4557
4558 BOOLVALUE   = "yes" / "no" / "true" / "false"
4559
4560 COLVALUE    = "{" OPTION-STRING "}"
4561
4562 DATEVALUE   = HOUR MINUTE [ SECOND ] / YEAR MONTH DAY /
4563             YEAR MONTH DAY HOUR MINUTE [ SECOND ]
4564
4565 YEAR        = 4DIGIT
4566
4567 MONTH       = "0" %x31-39 / "10" / "11" / "12"
4568
4569 DAY         = %x30-32 DIGIT / "1" DIGIT / "2" DIGIT / "30" / "31"
4570
4571 HOUR        = %x30-31 DIGIT / "1" DIGIT / "20" / "21" / "22" / "23"
4572
4573 MINUTE      = %x30-35 DIGIT
4574
4575 SECOND      = %x30-35 DIGIT
4576
4577 NUMBEVALUE  = 1*DIGIT / "-" 1*DIGIT / "+" 1*DIGIT
4578
4579 QUOTEDVALUE = DQUOTE *QUOTEDCHAR DQUOTE / SQUOTE *QUOTEDCHAR SQUOTE
4580
4581 QUOTEDCHAR  = %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
4582             %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
4583             %x5D-7E / %xA0-FF
4584
4585 OCTALDIGIT  = %x30-37
4586
4587 RANGEVALUE  = 1*DIGIT "-" 1*DIGIT
4588
4589 RESVALUE    = 1*DIGIT [ "x" 1*DIGIT ] ("dpi" / "dpc")
4590
4591 STRINGVALUE = 1*STRINGCHAR
4592
4593 STRINGCHAR  = %x5C %x20 / %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
4594             %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
4595             %x5D-7E / %xA0-FF
4596
4597 SQUOTE      = %x27
4598
4599 WC          = %x09 / %x0A / %x20
4600
```

4601 A.2. Examples

4602 The following example strings illustrate the format of text options:

4603 Boolean Attributes:

```
4604 foo
4605 nofoo
4606 foo=false
4607 foo=true
4608 foo=no
4609 foo=yes
4610
```

4611 **Collection Attributes:**
4612 media-col={media-size={x-dimension=123 y-dimension=456}}
4613

4614 **Integer Attributes:**
4615 copies=123
4616 hue=-123
4617

4618 **String Attributes:**
4619 job-sheets=standard
4620 job-sheets=standard, standard
4621 media=na-custom-foo.8000-10000
4622 job-name=John\'s\ Really\040Nice\ Document
4623

4624 **String Attributes (quoted):**
4625 job-name="John\'s Really Nice Document"
4626 document-name='Another \'Word\042 document.doc'
4627

4628 **Range Attributes:**
4629 page-ranges=1-5
4630 page-ranges=1-2,5-6,101-120
4631

4632 **Date Attributes:**
4633 job-hold-until-datetime=1234
4634 job-hold-until-datetime=123456
4635 job-hold-until-datetime=20020904
4636 job-hold-until-datetime=200209041234
4637 job-hold-until-datetime=20020904123456
4638

4639 **Resolution Attributes:**
4640 resolution=360dpi
4641 resolution=720x360dpi
4642 resolution=1000dpc
4643

4644 **Multiple Attributes:**
4645 job-sheets=standard page-ranges=1-2,5-6,101-120 resolution=360dpi
4646

4647 **Appendix B. References**

4648 **B.1. Internet Printing Protocol (IPP)**

4649 IETF RFCs can be obtained from "<http://www.rfc-editor.org/rfcsearch.html>".
4650 Other IPP documents can be obtained from
4651 "<http://www.pwg.org/ipp/index.html>" and
4652 "ftp://ftp.pwg.org/pub/pwg/ipp/new_XXX/".

4653 [RFC2911] T. Hastings, R. Herriot, R. deBry, S. Isaacson, and P. Powell August 1998
4654 *Internet Printing Protocol/1.1: Model and Semantics* (Obsoletes 2566)

4655 [RFC3196] T. Hastings, H. Holst, C. Kugler, C. Manros, and P. Zehler November
4656 2001 *Internet Printing Protocol/1.1: Implementor's Guide*

4657 [RFC3380] T. Hastings, R. Herriot, C. Kugler, and H. Lewis September 2002 *Internet*
4658 *Printing Protocol (IPP): Job and Printer Set Operations*

4659 [RFC3381] T. Hastings, H. Lewis, and R. Bergman September 2002 *Internet Printing*
4660 *Protocol (IPP): Job Progress Attributes*

4661 [RFC3382] R. deBry, T. Hastings, R. Herriot, K. Ocke, and P. Zehler September 2002
4662 *Internet Printing Protocol (IPP): The 'collection' attribute syntax*

4663 [5100.2] T. Hastings and R. Bergman IEEE-ISTO 5100.2 February 2001 *Internet*
4664 *Printing Protocol (IPP): output-bin attribute extension*

4665 [5100.3] T. Hastings and K. Ocke IEEE-ISTO 5100.3 February 2001 *Internet Printing*
4666 *Protocol (IPP): Production Printing Attributes*

4667 [5100.4] R. Herriot and K. Ocke IEEE-ISTO 5100.4 February 2001 *Internet Printing*
4668 *Protocol (IPP): Override Attributes for Documents and Pages*

4669 [5101.1] T. Hastings and D. Fullman IEEE-ISTO 5101.1 February 2001 *Internet*
4670 *Printing Protocol (IPP): finishings attribute values extension*

4671 [ops-set2] C. Kugler, T. Hastings, and H. Lewis July 2001 *Internet Printing Protocol*
4672 *(IPP): Job and Printer Administrative Operations*

4673 **B.2. Job Ticket**

4674 [jdf] CIP4 Organization April 2002 *Job Definition Format (JDF) Specification Version*
4675 *1.1*

4676 **B.3. Printer Working Group (PWG)**

4677 [PWGSemMod] P. Zehler and Albright September 2002 *Printer Working Group*
4678 *(PWG): Semantic Model*

4679 **B.4. Other**

4680 [RFC1738] T. Berners-Lee, L. Masinter, and M. McCahill December 1994 *Uniform*
4681 *Resource Locators (URL)* (Updated by RFC1808, RFC2368, RFC2396)

4682 [RFC2234] D. Crocker and P. Overell November 1997 *Augmented BNF for Syntax*
4683 *Specifications: ABNF*

4684 [RFC2396] T. Berners-Lee, R. Fielding, and L. Masinter August 1998 *Uniform*
4685 *Resource Locators (URL): Generic Syntax* (Updates RFC1808, RFC1738)

4686 **Appendix C. Change History**

4687 **Version 0.7 (October 18, 2002)**

4688

- 4689 • Added `attr_delim` argument to `papiAttributeListToString` and made new-line
4690 (`"\n"`) an allowed attribute delimiter on input to `papiAttributeListFromString`.
- 4691 • Added "Semantics Reference" subsections to functions.
- 4692 • Added to References: [5101.1], [RFC3196], and URIs for obtaining IPP
4693 documents.
- 4694 • Added `PAPI_JOB_TICKET_NOT_SUPPORTED` status code.
- 4695 • Added "Globalization" section in the "Print System Model" chapter.
- 4696 • Changed definition and usage of returned value from
4697 `papiAttributeListGetValue`. Also clarified what happens to output values when a
4698 `papiAttributeListGet*` call has an error.
- 4699 • Clarified descriptions of `papiPrinterGetAttributeList` and
4700 `papiJobGetAttributeList`.
- 4701 • Changed buffer length arguments from `int` to `size_t`.
- 4702 • Clarified that `papiServiceDestroy` must always be called after a call to
4703 `papiServiceCreate`.
- 4704 • Removed `attributes-charset`, `attributes-natural-language`, and `job-printer-up-time`
4705 from the "Required Job Attributes" (they may be hidden inside the PAPI
4706 implementation).
- 4707 • Clarified result of passing both attributes and a job ticket on all the job
4708 submission functions.
- 4709 • Miscellaneous wording and typo corrections.

4710

4711 **Version 0.6 (September 20, 2002)**

4712

- 4713 • Made explanation of `requested_attrs` in `papiPrintersList` the same as it is for
4714 `papiPrinterQuery`.
- 4715 • Moved `units` argument on `papiAttributeListAddResolution` to the end of the
4716 argument list to match the corresponding `get` function.
- 4717 • Added `papiAttributeListAddCollection` and `papiAttributeListGetCollection`.
- 4718 • Removed unneeded extra level of indirection from `attrs` argument to
4719 `papiAttributeListGet*` functions. Also made the `attrs` argument `const`.
- 4720 • Added note to "Conventions" section that strings are assumed to be UTF-8
4721 encoded.
- 4722 • Added `papiAttributeListFromString` and `papiAttributeListToString` functions,
4723 along with a new appendix defining the string format syntax.
- 4724 • Added `papiJobSubmitByReference`, `papiJobStreamOpen`, `papiJobStreamWrite`,
4725 and `papiJobStreamClose` functions.
- 4726 • Added short "Document" section in the "Print System Model" chapter.

- 4727 • Added explanation of how multiple files specified in the papiJobSubmit
- 4728 file_names argument are handled by the print system.
- 4729 • Changed papi_job_ticket_t "uri" field to "file_name" and added explanation text.
- 4730 • Added explanation of implementation option for merging papiJobSubmit
- 4731 attributes with job_ticket argument.
- 4732 • Added "References" appendix.
- 4733 • Added "IPP Attribute Type Mapping" appendix.
- 4734 • Added "PWG" job ticket format to papi_jt_format_t.
- 4735 • Miscellaneous wording and typo corrections.

4736

4737

Version 0.5 (August 30, 2002)

4738

- 4739 • Added job_attrs argument to papiPrinterQuery to support more accurate query
- 4740 of printer capabilities.
- 4741 • Added management functions papiAttributeDelete, papiJobModify, and
- 4742 papiPrinterModify.
- 4743 • Added functions papiAttributeListGetValue, papiAttributeListGetString,
- 4744 papiAttributeListGetInteger, etc.
- 4745 • Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent
- 4746 with the naming convention (first word after "papi" is the object being operated
- 4747 upon).
- 4748 • Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.
- 4749 • Made description of authentication more implementation-independent.
- 4750 • Added reference to IPP attributes summary document.
- 4751 • Added result argument to papiPrinterPurgeJobs.
- 4752 • Added "collection attribute" support (PAPI_COLLECTION type).
- 4753 • Changed boolean values to consistently use char. Added PAPI_FALSE and
- 4754 PAPI_TRUE enum values.

4755

4756

Version 0.4 (July 19, 2002)

4757

- 4758 • Made papi_job_t and papi_printer_t opaque handles and added "get" functions
- 4759 to access the associated information (papiPrinterGetAttributeList,
- 4760 papiJobGetAttributeList, papiJobGetId, papiJobGetPrinterName,
- 4761 papiJobGetJobTicket).
- 4762 • Removed variable length argument lists from attribute add functions.
- 4763 • Changed order and name of flag value passed to attribute add functions.
- 4764 • Eliminated indirection in date/time value passed to papiAttributeAddDatetime.
- 4765 • Added message argument to papiPrinterPause.

4766

4767

Version 0.3 (June 24, 2002)

4768

4769

- Converted to DocBook format from Microsoft Word

4770

- Major rewrite, including:

4771

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

4772

- Changed fixed length strings to pointers in numerous structures/sections

4773

- Redefined attribute/value structures and associated API descriptions

4774

- Changed list/query functions to return "objects"

4775

- Rewrote "Attributes API" chapter

4776

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

4777

4778

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

4779

4780

4781

4782

Version 0.2 (April 17, 2002)

4783

4784

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

4785

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

4786

4787

- Added "short_name" field in "Object Identification" section

4788

- Added "Job Ticket (papi_job_ticket_t)" section

4789

- Added papiPrinterPause

4790

- Added papiPrinterResume

4791

- Added papiPurgeJobs

4792

- Added optional job_ticket argument to papiJobSubmit

4793

- Added optional passing of filenames by URI to papiJobSubmit

4794

- Added papiHoldJob

4795

- Added papiReleaseJob

4796

- Added papiRestartJob

4797

4798

Version 0.1 (April 3, 2002)

4799

4800

- Original draft version

4801

4802

4803

4804

Appendix C. Change History

4805

4806

<i>End of Document</i>
