

1

2

Open Standard Print API (PAPI)

3

Version 0.5 (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 0.5 (DRAFT)**

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

14 Version 0.5 (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 Table of Contents

19	1. Introduction.....	1
20	2. Print System Model	2
21	2.1. Introduction.....	2
22	2.2. Model.....	2
23	2.2.1. Print Service	2
24	2.2.2. Printer	2
25	2.2.3. Job.....	3
26	2.3. Security.....	3
27	2.3.1. Authentication	3
28	2.3.2. Authorization.....	3
29	2.3.3. Encryption.....	3
30	3. Common Structures	4
31	3.1. Conventions.....	4
32	3.2. Service Object (papi_service_t)	4
33	3.3. Attributes and Values	4
34	3.4. Job Object (papi_job_t)	5
35	3.5. Printer Object (papi_printer_t).....	5
36	3.6. Job Ticket (papi_job_ticket_t).....	5
37	3.7. Status (papi_status_t)	6
38	3.8. List Filter (papi_filter_t).....	7
39	4. Service API	8
40	4.1. papiServiceCreate	8
41	4.2. papiServiceDestroy.....	9
42	4.3. papiServiceSetUsername	10
43	4.4. papiServiceSetPassword	12
44	4.5. papiServiceSetEncryption.....	13
45	4.6. papiServiceSetAuthCB.....	14
46	4.7. papiServiceSetAppData	15
47	4.8. papiServiceGetServicename.....	16
48	4.9. papiServiceGetUsername	17
49	4.10. papiServiceGetPassword	18
50	4.11. papiServiceGetEncryption.....	19
51	4.12. papiServiceGetAppData	19
52	4.13. papiServiceGetStatusMessage	20
53	5. Printer API.....	22
54	5.1. Usage	22
55	5.2. papiPrintersList.....	22
56	5.3. papiPrinterQuery.....	24
57	5.4. papiPrinterModify	26
58	5.5. papiPrinterPause.....	27
59	5.6. papiPrinterResume	29
60	5.7. papiPrinterPurgeJobs	30
61	5.8. papiPrinterListJobs	31
62	5.9. papiPrinterGetAttributeList.....	33
63	5.10. papiPrinterFree	34
64	5.11. papiPrinterListFree.....	35
65	6. Attributes API.....	37
66	6.1. papiAttributeListAdd	37

67	6.2. papiAttributeListAddString.....	38
68	6.3. papiAttributeListAddInteger.....	39
69	6.4. papiAttributeListAddBoolean.....	40
70	6.5. papiAttributeListAddRange.....	42
71	6.6. papiAttributeListAddResolution.....	43
72	6.7. papiAttributeListAddDatetime.....	44
73	6.8. papiAttributeDelete.....	46
74	6.9. papiAttributeListGetValue.....	47
75	6.10. papiAttributeListGetString.....	48
76	6.11. papiAttributeListGetInteger.....	49
77	6.12. papiAttributeListGetBoolean.....	50
78	6.13. papiAttributeListGetRange.....	51
79	6.14. papiAttributeListGetResolution.....	53
80	6.15. papiAttributeListGetDatetime.....	54
81	6.16. papiAttributeListFree.....	55
82	6.17. papiAttributeListFind.....	56
83	6.18. papiAttributeListGetNext.....	57
84	7. Job API.....	59
85	7.1. papiJobSubmit.....	59
86	7.2. papiJobValidate.....	60
87	7.3. papiJobQuery.....	62
88	7.4. papiJobModify.....	64
89	7.5. papiJobCancel.....	65
90	7.6. papiJobHold.....	66
91	7.7. papiJobRelease.....	68
92	7.8. papiJobRestart.....	69
93	7.9. papiJobGetAttributeList.....	70
94	7.10. papiJobGetPrinterName.....	72
95	7.11. papiJobGetId.....	72
96	7.12. papiJobGetJobTicket.....	73
97	7.13. papiJobFree.....	74
98	7.14. papiJobListFree.....	75
99	8. Miscellaneous API.....	77
100	8.1. papiStatusString.....	77
101	9. Attributes.....	78
102	9.1. Extension Attributes.....	78
103	9.1.1. job-ticket-formats-supported.....	78
104	9.2. Required Job Attributes.....	78
105	9.3. Required Printer Attributes.....	78
106	A. Change History.....	80

107 **Chapter 1. Introduction**

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

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

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

117

118

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

121 **Chapter 2. Print System Model**

122 **2.1. Introduction**

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

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

- 129 • Object Definition: What objects are part of the model?
- 130 • Object Naming: How is each object identified/named?
- 131 • Object Relationships: What are the associations and relationships between the
132 objects?

133

134 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	

135

136

137 **2.2. Model**

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

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

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

148 **2.2.1. Print Service**

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

153 **2.2.2. Printer**

154 Printer objects are the target of print job requests. A printer object may represent an
155 actual printer (if the printer itself supports PAPI), an object in a server representing
156 an actual printer, or an abstract object in a server (perhaps representing a pool or
157 class of printers). Printer objects are identified via one or more names which may be
158 short, local names (such as "prtr1") or longer global names (such as a URI like
159 "http://printserv.mycompany.com:631/printers/prtr1"). The PAPI implementation

160 may detect and map short names to long global names in an implementation-
161 specific way.

162 **2.2.3. Job**

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

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

169 **2.3. Security**

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

172 **2.3.1. Authentication**

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

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

179 **2.3.2. Authorization**

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

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

188 **2.3.3. Encryption**

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

194 Chapter 3. Common Structures

195 3.1. Conventions

196

197 • All "char*" variables and fields are pointers to standard C/C++ NULL-terminated
198 strings.

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

202

203 3.2. Service Object (papi_service_t)

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

207
208

```
typedef void* papi_service_t;
```

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

211
212
213
214
215
216
217
218

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

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

221 3.3. Attributes and Values

222 These are the structures defining how attributes and values are passed to and from
223 PAPI.

224
225
226
227
228
229
230
231
232
233
234
235

```
/* Attribute Type */  
typedef enum  
{  
    PAPI_STRING,  
    PAPI_INTEGER,  
    PAPI_BOOLEAN,  
    PAPI_RANGE,  
    PAPI_RESOLUTION,  
    PAPI_DATETIME,  
    PAPI_COLLECTION  
} papi_attribute_value_type_t;
```

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

237
238
239
240
241
242
243

```
/* Resolution units */  
typedef enum  
{  
    PAPI_RES_PER_INCH = 3,  
    PAPI_RES_PER_CM  
} papi_res_t;
```

244
245
246

```
/* Boolean values */  
enum  
{
```



```

247     PAPI_FALSE = 0,
248     PAPI_TRUE = 1
249 };
250
251 struct papi_attribute_str;
252
253 /* Attribute Value */
254 typedef union
255 {
256     char* string; /* PAPI_STRING value */
257
258     int integer; /* PAPI_INTEGER value */
259
260     char boolean; /* PAPI_BOOLEAN value */
261
262     struct /* PAPI_RANGE value */
263     {
264         int lower;
265         int upper;
266     } range;
267
268     struct /* PAPI_RESOLUTION value */
269     {
270         int xres;
271         int yres;
272         papi_res_t units;
273     } resolution;
274
275     time_t datetime; /* PAPI_DATETIME value */
276
277     struct papi_attribute_str**
278     collection; /* PAPI_COLLECTION value */
279 } papi_attribute_value_t;
280
281 /* Attribute and Values */
282 typedef struct papi_attribute_str
283 {
284     char* name; /* attribute name */
285     papi_attribute_value_type_t type; /* type of values */
286     papi_attribute_value_t** values; /* list of values */
287 } papi_attribute_t;
288
289 /* Attribute add flags */
290 #define PAPI_ATTR_APPEND 0x0001 /* Add values to attr */
291 #define PAPI_ATTR_REPLACE 0x0002 /* Delete existing
292 values then add new ones */
293 #define PAPI_ATTR_EXCL 0x0004 /* Fail if attr exists */
294

```

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

296 3.4. Job Object (papi_job_t)

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

301 3.5. Printer Object (papi_printer_t)

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

306 3.6. Job Ticket (papi_job_ticket_t)

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

```

311     /* Job Ticket Format */
312     typedef enum
313     {
314         PAPI_JT_FORMAT_JDF = 0,      /* Job Definition Format */
315     } papi_jt_format_t;
316

```

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

```

318     /* Job Ticket */
319     typedef struct papi_job_ticket_s
320     {
321         papi_jt_format_t format;      /* Format of job ticket */
322         char* ticket_data;           /* Buffer containing the job
323                                         ticket data. If NULL,
324                                         uri must be specified */
325         char* uri;                   /* URI of the file containing
326                                         the job ticket data. If
327                                         ticket_data is specified, then
328                                         uri is ignored. */
329     } papi_job_ticket_t;
330

```

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

332 3.7. Status (papi_status_t)

```

333     typedef enum
334     {
335         PAPI_OK = 0x0000,
336         PAPI_OK_SUBST,
337         PAPI_OK_CONFLICT,
338         PAPI_OK_IGNORED_SUBSCRIPTIONS,
339         PAPI_OK_IGNORED_NOTIFICATIONS,
340         PAPI_OK_TOO_MANY_EVENTS,
341         PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
342         PAPI_REDIRECTION_OTHER_SITE = 0x300,
343         PAPI_BAD_REQUEST = 0x0400,
344         PAPI_FORBIDDEN,
345         PAPI_NOT_AUTHENTICATED,
346         PAPI_NOT_AUTHORIZED,
347         PAPI_NOT_POSSIBLE,
348         PAPI_TIMEOUT,
349         PAPI_NOT_FOUND,
350         PAPI_GONE,
351         PAPI_REQUEST_ENTITY,
352         PAPI_REQUEST_VALUE,
353         PAPI_DOCUMENT_FORMAT,
354         PAPI_ATTRIBUTES,
355         PAPI_URI_SCHEME,
356         PAPI_CHARSET,
357         PAPI_CONFLICT,
358         PAPI_COMPRESSION_NOT_SUPPORTED,
359         PAPI_COMPRESSION_ERROR,
360         PAPI_DOCUMENT_FORMAT_ERROR,
361         PAPI_DOCUMENT_ACCESS_ERROR,
362         PAPI_ATTRIBUTES_NOT_SETTABLE,
363         PAPI_IGNORED_ALL_SUBSCRIPTIONS,
364         PAPI_TOO_MANY_SUBSCRIPTIONS,
365         PAPI_IGNORED_ALL_NOTIFICATIONS,
366         PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
367         PAPI_INTERNAL_ERROR = 0x0500,
368         PAPI_OPERATION_NOT_SUPPORTED,
369         PAPI_SERVICE_UNAVAILABLE,
370         PAPI_VERSION_NOT_SUPPORTED,
371         PAPI_DEVICE_ERROR,
372         PAPI_TEMPORARY_ERROR,
373         PAPI_NOT_ACCEPTING,
374         PAPI_PRINTER_BUSY,
375         PAPI_ERROR_JOB_CANCELLED,
376         PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
377         PAPI_PRINTER_IS_DEACTIVATED,
378         PAPI_BAD_ARGUMENT
379     } papi_status_t;
380

```

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

384 **3.8. List Filter (papi_filter_t)**

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

```

388 typedef enum
389 {
390     PAPI_FILTER_BITMASK = 0
391     /* future filter types may be added here */
392 } papi_filter_type_t;
393
394 typedef struct
395 {
396     papi_filter_type_t type; /* Type of filter specified */
397
398     union
399     {
400         unsigned int mask; /* PAPI_FILTER_BITMASK */
401
402         /* future filter types may be added here */
403     } u;
404 } papi_filter_t;
405

```

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

```

408 enum
409 {
410     PAPI_PRINTER_LOCAL = 0x0000, /* Local printer or class */
411     PAPI_PRINTER_CLASS = 0x0001, /* Printer class */
412     PAPI_PRINTER_REMOTE = 0x0002, /* Remote printer or class */
413     PAPI_PRINTER_BW = 0x0004, /* Can do B&W printing */
414     PAPI_PRINTER_COLOR = 0x0008, /* Can do color printing */
415     PAPI_PRINTER_DUPLEX = 0x0010, /* Can do duplexing */
416     PAPI_PRINTER_STAPLE = 0x0020, /* Can staple output */
417     PAPI_PRINTER_COPIES = 0x0040, /* Can do copies */
418     PAPI_PRINTER_COLLATE = 0x0080, /* Can collage copies */
419     PAPI_PRINTER_PUNCH = 0x0100, /* Can punch output */
420     PAPI_PRINTER_COVER = 0x0200, /* Can cover output */
421     PAPI_PRINTER_BIND = 0x0400, /* Can bind output */
422     PAPI_PRINTER_SORT = 0x0800, /* Can sort output */
423     PAPI_PRINTER_SMALL = 0x1000, /* Can do Letter/Legal/A4 */
424     PAPI_PRINTER_MEDIUM = 0x2000, /* Can do Tabloid/B/C/A3/A2 */
425     PAPI_PRINTER_LARGE = 0x4000, /* Can do D/E/A1/A0 */
426     PAPI_PRINTER_VARIABLE = 0x8000, /* Can do variable sizes */
427     PAPI_PRINTER_IMPLICIT = 0x10000, /* Implicit class */
428     PAPI_PRINTER_DEFAULT = 0x20000, /* Default printer on network */
429     PAPI_PRINTER_OPTIONS = 0xffff, /* ~(CLASS | REMOTE | IMPLICIT) */
430 };
431

```

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

433 Chapter 4. Service API

434 4.1. papiServiceCreate

435 Description

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

440 Syntax

441

```
442 papi_status_t papiServiceCreate(  
443     papi_service_t*      handle,  
444     const char*          service_name,  
445     const char*          user_name,  
446     const char*          password,  
447     const int (*authCB)(papi_service_t svc),  
448     const papi_encryption_t encryption,  
449     void*                app_data );  
450
```

451

452 Inputs

453

454 service_name

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

459 user_name

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

463 password

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

466 authCB

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

472 encryption

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

474 app_data

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

477

478 **Outputs**

479

480 handle

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

484

485 **Returns**

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

488 **Example**

489

```

490 #include "papi.h"
491
492 papi_status_t status;
493 papi_service_t handle = NULL;
494 const char* service_name = "ipp:/printserv:631";
495 const char* user_name = "pappy";
496 const char* password = "goober";
497 ...
498 status = papiServiceCreate(&handle,
499                             service_name,
500                             user_name,
501                             password,
502                             NULL,
503                             PAPI_ENCRYPT_IF_REQUESTED,
504                             NULL);
505
506 if (status != PAPI_OK)
507 {
508     /* handle the error */
509     fprintf(stderr, "papiServiceCreate failed: %s\n",
510             papiStatusString(status));
511     if (handle != NULL)
512     {
513         fprintf(stderr, "    details: %s\n",
514                 papiServiceGetStatusMessage(handle));
515     }
516     ...
517 }
518 ...
519 papiServiceDestroy(handle);

```

520

521 **See Also**

522 papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUsername,
523 papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB

524 4.2. papiServiceDestroy

525 **Description**

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

529 **Syntax**

530

```
531     void papiServiceDestroy(
532         papi_service_t handle );
533
```

534

535 **Inputs**

536

537 handle

538 The print service handle to be destroyed.

539

540 **Outputs**

541 none

542 **Returns**

543 none

544 **Example**

545

```
546     #include "papi.h"
547
548     papi_status_t status;
549     papi_service_t handle = NULL;
550     const char* service_name = "ipp://printserv:631";
551     const char* user_name = "pappy";
552     const char* password = "goober";
553     ...
554     status = papiServiceCreate(&handle,
555                               service_name,
556                               user_name,
557                               password,
558                               NULL,
559                               PAPI_ENCRYPT_IF_REQUESTED,
560                               NULL);
561
562     if (status != PAPI_OK)
563     {
564         /* handle the error */
565         ...
566     }
567     ...
568     papiServiceDestroy(handle);
```

569

570 **See Also**

571 papiServiceCreate

572 **4.3. papiServiceSetUsername**573 **Description**

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

577 **Syntax**

578

```

579     papi_status_t papiServiceSetUsername (
580         papi_service_t handle,
581         const char* user_name );
582

```

583

584 **Inputs**

585

586 **handle**

587 Handle to the print service to update.

588 **user_name**

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

592

593 **Outputs**

594 handle is updated.

595 **Returns**

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

598 **Example**

599

```

600     #include "papi.h"
601
602     papi_status_t status;
603     papi_service_t handle = NULL;
604     const char* user_name = "pappy";
605     ...
606     status = papiServiceCreate(&handle,
607                               NULL,
608                               NULL,
609                               NULL,
610                               NULL,
611                               PAPI_ENCRYPT_IF_REQUESTED,
612                               NULL);
613
614     if (status != PAPI_OK)
615     {
616         /* handle the error */
617         ...
618     }
619
620     status = papiServiceSetUsername(handle, user_name);
621     if (status != PAPI_OK)
622     {
623         /* handle the error */
624         fprintf(stderr, "papiServiceSetUsername failed: %s\n",
625               papiServiceGetStatusMessage(handle));
626         ...
627     }
628     ...
629     papiServiceDestroy(handle);

```

630

631 **See Also**

632 papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

633 **4.4. papiServiceSetPassword**634 **Description**

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

638 **Syntax**

639

```
640 papi_status_t papiServiceSetPassword(
641     papi_service_t handle,
642     const char* password );
643
```

644

645 **Inputs**

646

647 handle

648 Handle to the print service to update.

649 password

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

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* password = "goober";
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 = papiServiceSetPassword(handle, password);
680 if (status != PAPI_OK)
681 {
682     /* handle the error */
683     fprintf(stderr, "papiServiceSetPassword failed: %s\n",
684         papiServiceGetStatusMessage(handle));
685     ...
686 }
```



```
687     papiServiceDestroy(handle);
688
```

689

690 **See Also**

691 papiServiceCreate, papiServiceSetUsername, papiServiceGetStatusMessage

692 4.5. papiServiceSetEncryption

693 **Description**

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

695 **Syntax**

696

```
697     papi_status_t papiServiceSetEncryption(
698         papi_service_t handle,
699         const papi_encryption_t encryption );
700
```

701

702 **Inputs**

703

704 handle

705 Handle to the print service to update.

706 encryption

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

708

709 **Outputs**

710 handle is updated.

711 **Returns**

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

714 **Example**

715

```
716     #include "papi.h"
717
718     papi_status_t status;
719     papi_service_t handle = NULL;
720     ...
721     status = papiServiceCreate(&handle,
722                               NULL,
723                               NULL,
724                               NULL,
725                               NULL,
726                               PAPI_ENCRYPT_IF_REQUESTED,
727                               NULL);
728
729     if (status != PAPI_OK)
730     {
731         /* handle the error */
732         ...
733     }
734
735     status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
736     if (status != PAPI_OK)
```

```

736     {
737         /* handle the error */
738         fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
739                papiServiceGetStatusMessage(handle));
740         ...
741     }
742     ...
743     papiServiceDestroy(handle);
744

```

745

746 **See Also**

747 papiServiceCreate, papiServiceGetStatusMessage

748 **4.6. papiServiceSetAuthCB**749 **Description**

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

752 **Syntax**

753

```

754 papi_status_t papiServiceSetAuthCB(
755     papi_service_t handle,
756     const int (*authCB)(papi_service_t svc) );
757

```

758

759 **Inputs**

760

761 handle

762 Handle to the print service to update.

763 authCB

764 Points to a callback function to be used in authenticating the user to the print
 765 service if no password was supplied (or user input is required). A NULL value
 766 indicates that no callback should be made. The callback function should return
 767 0 if the request is to be cancelled and non-zero if new authentication
 768 information has been set.

769

770 **Outputs**

771 handle is updated.

772 **Returns**

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

775 **Example**

776

```

777 #include "papi.h"
778
779 extern int get_password(papi_service_t handle);
780 papi_status_t status;
781 papi_service_t handle = NULL;

```

```

782     ...
783     status = papiServiceCreate(&handle,
784                               NULL,
785                               NULL,
786                               NULL,
787                               NULL,
788                               PAPI_ENCRYPT_IF_REQUESTED,
789                               NULL);
790
791     if (status != PAPI_OK)
792     {
793         /* handle the error */
794         ...
795     }
796
797     status = papiServiceSetAuthCB(handle, get_password);
798     if (status != PAPI_OK)
799     {
800         /* handle the error */
801         fprintf(stderr, "papiServiceSetAuthCB failed: %s\n",
802                papiServiceGetStatusMessage(handle));
803         ...
804     }
805     ...
806     papiServiceDestroy(handle);

```

807

808 **See Also**

809 papiServiceCreate, papiServiceGetStatusMessage

810 **4.7. papiServiceSetAppData**811 **Description**

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

815 **Syntax**

816

```

817 papi_status_t papiServiceSetAppData (
818     papi_service_t handle,
819     const void* app_data );
820

```

821

822 **Inputs**

823

824 handle

825 Handle to the print service to update.

826 app_data

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

829

830 **Outputs**

831 handle is updated.

832

Returns

833

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

834

835

Example

836

837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867

```
#include "papi.h"

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

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

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

868

869

See Also

870

papiServiceCreate, papiServiceGetStatusMessage

871

4.8. papiServiceGetServicename

872

Description

873

Get the service name associated with the print service handle.

874

Syntax

875

876

877

878

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

879

880

Inputs

881

882

handle

883

Handle to the print service.

884

885

Outputs

886

none

887

Returns

888

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

889

Example

890

```

891 #include "papi.h"
892
893 papi_status_t status;
894 papi_service_t handle = NULL;
895 char* service_name = NULL;
896 ...
897 service_name = papiServiceGetServicename(handle);
898 if (service_name != NULL)
899 {
900     /* use the returned name */
901     ...
902 }
903 ...
904 papiServiceDestroy(handle);
905

```

906

907

See Also

908

papiServiceCreate

4.9. papiServiceGetUsername

910

Description

911

Get the user name associated with the print service handle.

912

Syntax

913

```

914 char* papiServiceGetUsername(
915     papi_service_t handle );
916

```

917

918

Inputs

919

920 handle

921

Handle to the print service.

922

923

Outputs

924

none

925

Returns

926

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

927

Example

928

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

944

945 **See Also**

946 papiServiceCreate, papiServiceSetUsername

947 **4.10. papiServiceGetPassword**

948 **Description**

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

950 **Syntax**

951

```
952         char* papiServiceGetPassword(
953             papi_service_t handle );
954
```

955

956 **Inputs**

957

958 handle

959 Handle to the print service.

960

961 **Outputs**

962 none

963 **Returns**

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

965 **Example**

966

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

982

983

See Also

984

papiServiceCreate, papiServiceSetPassword

985 **4.11. papiServiceGetEncryption**

986

Description

987

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

988

Syntax

989

990

```
papi_encryption_t papiServiceGetEncryption(
991     papi_service_t handle );
992
```

992

993

994

Inputs

995

996

handle

997

Handle to the print service.

998

999

Outputs

1000

none

1001

Returns

1002

The type of encryption associated with the print service handle.

1003

Example

1004

1005

```
#include "papi.h"
1006
1007 papi_status_t status;
1008 papi_service_t handle = NULL;
1009 papi_encryption_t encryption;
1010 ...
1011 encryption = papiServiceGetEncryption(handle);
1012 /* use the returned encryption value */
1013 ...
1014 papiServiceDestroy(handle);
1015
```

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

See Also

1018

papiServiceCreate, papiServiceSetEncryption

1019

4.12. papiServiceGetAppData

1020

Description

1021

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

1022

handle.

1023 **Syntax**

1024

```
1025           void* papiServiceGetAppData(
1026                     papi_service_t handle );
1027
```

1028

1029 **Inputs**

1030

1031 handle

1032 Handle to the print service.

1033

1034 **Outputs**

1035 none

1036 **Returns**

1037 A pointer to the application-specific data associated with the print service handle.

1038 **Example**

1039

```
1040           #include "papi.h"
1041
1042           papi_status_t status;
1043           papi_service_t handle = NULL;
1044           char* app_data = NULL;
1045           ...
1046           app_data = (char*)papiServiceGetAppData(handle);
1047           if (app_data != NULL)
1048           {
1049               /* use the returned application data */
1050               ...
1051           }
1052           ...
1053           papiServiceDestroy(handle);
1054
```

1055

1056 **See Also**

1057 papiServiceCreate, papiServiceSetAppData

1058 **4.13. papiServiceGetStatusMessage**

1059 **Description**

1060 Get the message associated with the status of the last operation performed. The
1061 status message returned from this function may be more detailed than the status
1062 message returned from papiStatusString (if the print service supports returning
1063 more detailed error messages).

1064 The returned message will be localized in the language of the submitter of the
1065 original operation.

1066 **Syntax**

1067

```
1068           const char* papiServiceGetStatusMessage(
```



```
1069         const papi_service_t handle );
1070
```

1071

1072 **Inputs**

1073

1074 handle

Handle to the print service.

1076

1077 **Outputs**

1078 none

1079 **Returns**

1080 Pointer to the message associated with the status of the last operation performed.

1081 **Example**

1082

```
1083 #include "papi.h"
1084
1085 papi_status_t status;
1086 papi_service_t handle = NULL;
1087 const char* user_name = "pappy";
1088 ...
1089 status = papiServiceCreate(&handle,
1090                             NULL,
1091                             NULL,
1092                             NULL,
1093                             NULL,
1094                             PAPI_ENCRYPT_IF_REQUESTED,
1095                             NULL);
1096
1097 if (status != PAPI_OK)
1098 {
1099     /* handle the error */
1100     ...
1101 }
1102
1103 status = papiServiceSetUsername(handle, user_name);
1104 if (status != PAPI_OK)
1105 {
1106     /* handle the error */
1107     fprintf(stderr, "papiServiceSetUsername failed: %s\n",
1108             papiServiceGetStatusMessage(handle));
1109     ...
1110 }
1111 ...
1112 papiServiceDestroy(handle);
```

1113

1114 **See Also**

1115 papiStatusString

1116 Chapter 5. Printer API

1117 5.1. Usage

1118 The papiPrinterQuery function queries all/some of the attributes of a printer
1119 object. It returns a list of printer attributes. A successful call to papiPrinterQuery is
1120 typically followed by code which examines and processes the returned attributes.
1121 The using program would then call papiPrinterFree to delete the returned results.

1122 Printers can be found via calls to papiPrintersList. A successful call to
1123 papiPrintersList is typically followed by code to iterate through the list of returned
1124 printers, possibly querying each (papiPrinterQuery) for further information (e.g. to
1125 restrict what printers get displayed for a particular user/request). The using
1126 program would then call papiPrinterListFree to free the returned results.

1127 5.2. papiPrintersList

1128 Description

1129 List all printers known by the print service which match the specified filter.

1130 Depending on the functionality of the target service's "printer directory", the
1131 returned list may be limited to only printers managed by a particular server or it
1132 may include printers managed by other servers.

1133 Syntax

```
1135 papi_status_t papiPrintersList(  
1136     papi_service_t handle,  
1137     const char* requested_attrs[],  
1138     const papi_filter_t* filter,  
1139     papi_printer_t** printers );  
1140
```

1142 Inputs

1144 handle

1145 Handle to the print service to use.

1146 requested_attrs

1147 (optional) NULL terminated array of attribute names to be queried. If NULL is
1148 passed then all available attributes should be returned.

1149 filter

1150 (optional) Pointer to a filter to limit the number of printers returned on the list
1151 request. See Section 3.8 for details. If NULL is passed then all known printers
1152 are listed.

1154 Outputs

1156 printers

1157 List of printer objects that matched the filter criteria.

1158

1159 **Returns**

1160 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1161 value is returned.

1162 **Example**

1163

```

1164 #include "papi.h"
1165
1166 int i;
1167 papi_status_t status;
1168 papi_service_t handle = NULL;
1169 const char* service_name = "ipp://printserv:631";
1170 const char* user_name = "pappy";
1171 const char* password = "goober";
1172 const char* req_attrs[] =
1173 {
1174     "printer-name",
1175     "printer-location",
1176     NULL
1177 };
1178 const papi_filter_t filter =
1179     PAPI_PRINTER_BW | PAPI_PRINTER_DUPLEX;
1180 papi_printer_t* printers = NULL;
1181 ...
1182 status = papiServiceCreate(&handle,
1183                             service_name,
1184                             user_name,
1185                             password,
1186                             NULL,
1187                             PAPI_ENCRYPT_IF_REQUESTED,
1188                             NULL);
1189
1190 if (status != PAPI_OK)
1191 {
1192     /* handle the error */
1193     ...
1194 }
1195
1196 status = papiPrinterList(handle,
1197                           req_attrs,
1198                           filter,
1199                           &printers);
1200
1201 if (status != PAPI_OK)
1202 {
1203     /* handle the error */
1204     fprintf(stderr, "papiPrinterList failed: %s\n",
1205             papiServiceGetStatusMessage(handle));
1206     ...
1207 }
1208
1209 if (printers != NULL)
1210 {
1211     for (i=0; printers[i] != NULL; i++)
1212     {
1213         /* process the printer object */
1214         ...
1215     }
1216     papiPrinterListFree(printers);
1217 }
1218
1219 papiServiceDestroy(handle);

```

1219

1220 **See Also**

1221 papiPrinterListFree, papiPrinterQuery

1222 **5.3. papiPrinterQuery**1223 **Description**

1224 Queries some or all the attributes of the specified printer object. This includes
 1225 attributes representing the capabilities of the printer, which the caller may use to
 1226 determine which print options to present to the user. How the attributes are
 1227 obtained (e.g. from a static database, from a dialog with the hardware, from a dialog
 1228 with a driver, etc.) is up to the implementer of the API and is beyond the scope of
 1229 this standard.

1230 This optionally includes "context" information which specifies job attributes in the
 1231 context of which the capabilities information is to be constructed.

1232 **Syntax**

1233

```

1234 papi_status_t papiPrinterQuery(
1235             papi_service_t   handle,
1236             const char*      name,
1237             const char*      requested_attrs[],
1238             const papi_attribute_t** job_attrs,
1239             papi_printer_t*  printer );
1240

```

1241

1242 **Inputs**

1243

1244 handle

1245 Handle to the print service to use.

1246 name

1247 The name or URI of the printer to query.

1248 requested_attrs

1249 (optional) NULL terminated array of attributes to be queried. If NULL is
 1250 passed then all attributes are queried. (NOTE: The printer may return more
 1251 attributes than you requested. This is merely an advisory request that may
 1252 reduce the amount of data returned if the printer/server supports it.)

1253 job_attrs

1254 (optional) NULL terminated array of job attributes in the context of which the
 1255 capabilities information is to be constructed. In other words, the returned
 1256 printer attributes represent the capabilities of the printer given that these
 1257 specified job attributes are requested. This allows for more accurate
 1258 information to be retrieved by the caller for a specific job (e.g. "if the job is
 1259 printed on A4 size media then duplex output is not available"). If NULL is
 1260 passed then the full capabilities of the printer are queried.

1261 Support for this argument is optional. If the underlying print system does not
 1262 have access to capabilities information bound by job context, then this
 1263 argument may be ignored. But if the calling application will be using the
 1264 returned information to build print job data, then it is always advisable to
 1265 specify the job context attributes. The more context information provided, the

1266 more accurate capabilities information is likely to be returned from the print
 1267 system.

1268

1269 **Outputs**

1270

1271 printer

Pointer to a printer object containing the requested attributes.

1272

1273

1274 **Returns**

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

1275

1276

1277 **Example**

1278

```

1279 #include "papi.h"
1280
1281 papi_status_t status;
1282 papi_service_t handle = NULL;
1283 const char* service_name = "ipp://printserv:631";
1284 const char* user_name = "pappy";
1285 const char* password = "goober";
1286 const char* printer_name = "my-printer";
1287 const char* req_attrs[] =
1288 {
1289     "printer-name",
1290     "printer-location",
1291     "printer-state",
1292     "printer-state-reasons",
1293     "printer-state-message",
1294     NULL
1295 };
1296 papi_attribute_t** job_attrs = NULL;
1297 papi_printer_t printer = NULL;
1298 ...
1299 status = papiServiceCreate(&handle,
1300                             service_name,
1301                             user_name,
1302                             password,
1303                             NULL,
1304                             PAPI_ENCRYPT_IF_REQUESTED,
1305                             NULL);
1306
1307 if (status != PAPI_OK)
1308 {
1309     /* handle the error */
1310     ...
1311 }
1312
1313 papiAttributeListAddString(&job_attrs,
1314                             PAPI_EXCL,
1315                             "media",
1316                             "legal");
1317
1318 status = papiPrinterQuery(handle,
1319                             printer_name,
1320                             req_attrs,
1321                             job_attrs,
1322                             &printer);
1323
1324 if (status != PAPI_OK)
1325 {
1326     /* handle the error */
1327     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1328             papiServiceGetStatusMessage(handle));
1329     ...
1330 }
1331
1332 if (printer != NULL)
1333 {
1334     /* process the printer object */
1335     ...
1336     papiPrinterFree(printer);
1337 }

```

```
1337     papiAttributeListFree(job_attrs);  
1338     papiServiceDestroy(handle);  
1339
```

1340

1341 **See Also**

1342 papiPrinterList, papiPrinterFree, papiPrinterModify

1343 5.4. papiPrinterModify

1344 Description

1345 Modifies some or all the attributes of the specified printer object.

1346 Syntax

1347

```
1348     papi_status_t papiPrinterModify(  
1349         papi_service_t    handle,  
1350         const char*       printer_name,  
1351         const papi_attribute_t** attrs,  
1352         papi_printer_t*   printer );  
1353
```

1354

1355 Inputs

1356

1357 handle

1358 Handle to the print service to use.

1359 printer_name

1360 Pointer to the name or URI of the printer to be modified.

1361 attrs

1362 Attributes to be modified. Any attributes not specified are left unchanged.

1363

1364 Outputs

1365

1366 printer

1367 The modified printer object.

1368

1369 Returns

1370 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1371 value is returned.

1372 Example

1373

```
1374     #include "papi.h"  
1375     papi_status_t status;
```

```

1377     papi_service_t handle = NULL;
1378     const char* printer_name = "my-printer";
1379     papi_printer_t printer = NULL;
1380     papi_attribute_t** attrs = NULL;
1381     ...
1382     status = papiServiceCreate(&handle,
1383                               NULL,
1384                               NULL,
1385                               NULL,
1386                               NULL,
1387                               PAPI_ENCRYPT_NEVER,
1388                               NULL);
1389
1390     if (status != PAPI_OK)
1391     {
1392         /* handle the error */
1393         ...
1394     }
1395
1396     papiAttributeListAddString(&attrs,
1397                               PAPI_EXCL,
1398                               "printer-location",
1399                               "Bldg 17/Room 234");
1400
1401     status = papiPrinterModify(handle,
1402                               printer_name,
1403                               attrs,
1404                               &printer);
1405
1406     if (status != PAPI_OK)
1407     {
1408         /* handle the error */
1409         fprintf(stderr, "papiPrinterModify failed: %s\n",
1410                papiServiceGetStatusMessage(handle));
1411         ...
1412     }
1413
1414     if (printer != NULL)
1415     {
1416         /* process the printer */
1417         ...
1418         papiPrinterFree(printer);
1419     }
1420
1421     papiServiceDestroy(handle);

```

1421

1422

See Also

1423

papiPrinterQuery, papiPrinterFree

1424

5.5. papiPrinterPause

1425

Description

1426

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.

1427

1428

1429

1430

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

1431

1432

Syntax

1433

1434

```

1435     papi_status_t papiPrinterPause(
1436         papi_service_t handle,
1437         const char* name,
1438         const char* message );

```

1438

1439

1440 **Inputs**

1441

1442 handle

1443 Handle to the print service to use.

1444 name

1445 The name or URI of the printer to operate on.

1446 message

1447 (optional) An explanatory message to be associated with the paused printer.
 1448 This message may be ignored if the underlying print system does not support
 1449 associating a message with a paused printer.

1450

1451 **Outputs**

1452 none

1453 **Returns**

1454 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1455 value is returned.

1456 **Example**

1457

```

1458           #include "papi.h"
1459
1460           papi_status_t status;
1461           papi_service_t handle = NULL;
1462           const char* service_name = "ipp://printserv:631";
1463           const char* user_name = "pappy";
1464           const char* password = "goober";
1465           const char* printer_name = "my-printer";
1466           ...
1467           status = papiServiceCreate(&handle,
1468                                    service_name,
1469                                    user_name,
1470                                    password,
1471                                    NULL,
1472                                    PAPI_ENCRYPT_IF_REQUESTED,
1473                                    NULL);
1474
1475           if (status != PAPI_OK)
1476           {
1477               /* handle the error */
1478               ...
1479           }
1480
1481           status = papiPrinterPause(handle, printer_name, NULL);
1482           if (status != PAPI_OK)
1483           {
1484               /* handle the error */
1485               fprintf(stderr, "papiPrinterPause failed: %s\n",
1486                        papiServiceGetStatusMessage(handle));
1487               ...
1488           }
1489           ...
1490           papiServiceDestroy(handle);

```

1491

1492 **See Also**

1493 papiPrinterResume

1494 **5.6. papiPrinterResume**1495 **Description**

1496 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the
 1497 effects of papiPrinterPause). This operation is optional and may not be supported
 1498 by all printers/servers, but it must be supported if papiPrinterPause is supported.

1499 **Syntax**

1500

```
1501 papi_status_t papiPrinterResume(
1502             papi_service_t   handle,
1503             const char*      name );
1504
```

1505

1506 **Inputs**

1507

1508 handle

Handle to the print service to use.

1509

1510 name

The name or URI of the printer to operate on.

1511

1512

1513 **Outputs**

1514 none

1515 **Returns**

1516 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1517 value is returned.

1518 **Example**

1519

```
1520 #include "papi.h"
1521
1522 papi_status_t status;
1523 papi_service_t handle = NULL;
1524 const char* service_name = "ipp://printserv:631";
1525 const char* user_name = "pappy";
1526 const char* password = "goober";
1527 const char* printer_name = "my-printer";
1528 ...
1529 status = papiServiceCreate(&handle,
1530                          service_name,
1531                          user_name,
1532                          password,
1533                          NULL,
1534                          PAPI_ENCRYPT_IF_REQUESTED,
1535                          NULL);
1536
1537 if (status != PAPI_OK)
1538 {
1539     /* handle the error */
1540     ...
1541 }
1542
1543 status = papiPrinterPause(handle, printer_name);
1544 if (status != PAPI_OK)
1545 {
1546     /* handle the error */
1547     fprintf(stderr, "papiPrinterPause failed: %s\n",
1548            papiServiceGetStatusMessage(handle));
1549 }
```

```

1548     ...
1549     }
1550     ...
1551     status = papiPrinterResume(handle, printer_name);
1552     if (status != PAPI_OK)
1553     {
1554         /* handle the error */
1555         fprintf(stderr, "papiPrinterResume failed: %s\n",
1556                papiServiceGetStatusMessage(handle));
1557         ...
1558     }
1559
1560     papiServiceDestroy(handle);
1561

```

1562

1563 **See Also**

1564 papiPrinterPause

1565 **5.7. papiPrinterPurgeJobs**1566 **Description**

1567 Remove all jobs from the specified printer object regardless of their states. This
 1568 includes removing jobs that have completed and are being kept for history (if any).
 1569 This operation is optional and may not be supported by all printers/servers.

1570 **Syntax**

1571

```

1572 papi_status_t papiPrinterPurgeJobs(
1573     papi_service_t handle,
1574     const char* name,
1575     papi_job_t** result);
1576

```

1577

1578 **Inputs**

1579

1580 handle

Handle to the print service to use.

1582 name

The name or URI of the printer to operate on.

1584

1585 **Outputs**

1586

1587 result

1588 (optional) Pointer to a list of purged jobs with the identifying information (job-
 1589 id/job-uri), success/fail, and possibly a detailed message. If NULL is passed
 1590 then no job list is returned. Support for the returned job list is optional and may
 1591 not be supported by all implementations (if not supported, the function
 1592 completes with PAPI_OK_SUBST but no list is returned).

1593 name

1594 The name or URI of the printer to operate on.

1595

1596 **Returns**

1597 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1598 value is returned.

1599 **Example**

1600

```

1601 #include "papi.h"
1602
1603 papi_status_t status;
1604 papi_service_t handle = NULL;
1605 const char* service_name = "ipp://printserv:631";
1606 const char* user_name = "pappy";
1607 const char* password = "goober";
1608 const char* printer_name = "my-printer";
1609 ...
1610 status = papiServiceCreate(&handle,
1611                             service_name,
1612                             user_name,
1613                             password,
1614                             NULL,
1615                             PAPI_ENCRYPT_IF_REQUESTED,
1616                             NULL);
1617
1618 if (status != PAPI_OK)
1619 {
1620     /* handle the error */
1621     ...
1622 }
1623
1624 status = papiPrinterPurgeJobs(handle, printer_name);
1625 if (status != PAPI_OK)
1626 {
1627     /* handle the error */
1628     fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
1629             papiServiceGetStatusMessage(handle));
1630     ...
1631 }
1632
1633 papiServiceDestroy(handle);

```

1634

1635 **See Also**

1636 papiJobCancel

1637 5.8. papiPrinterListJobs

1638 **Description**

1639 List print job(s) associated with the specified printer.

1640 **Syntax**

1641

```

1642 papi_status_t papiPrinterListJobs(
1643     papi_service_t handle,
1644     const char* printer,
1645     const char* requested_attrs[],
1646     const int type_mask,
1647     const int max_num_jobs,
1648     papi_job_t** jobs );
1649

```

1650

1651 **Inputs**

1652

1653 handle

1654 Handle to the print service to use.

1655 requested_attrs

1656 (optional) NULL terminated array of attributes to be queried. If NULL is
 1657 passed then all available attributes are queried. (NOTE: The printer may return
 1658 more attributes than you requested. This is merely an advisory request that
 1659 may reduce the amount of data returned if the printer/server supports it.)

1660 type_mask

1661 A bit mask which determines what jobs will get returned. The following
 1662 constants can be bitwise-OR-ed together to select which types of jobs to list:

```

1663           #define PAPI_LIST_JOBS_OTHERS        0x0001 /* return jobs other than
1664                                                    those submitted by the
1665                                                    user name assoc with
1666                                                    the handle */
1667           #define PAPI_LIST_JOBS_COMPLETED     0x0002 /* return completed jobs */
1668           #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
1669                                                    jobs */
1670           #define PAPI_LIST_JOBS_ALL           0xFFFF /* return all jobs */
1671

```

1672

1673 max_num_jobs

1674 Limit to the number of jobs returned. If 0 is passed, then there is no limit on
 1675 the number of jobs which may be returned.

1676

1677 **Outputs**

1678

1679 jobs

1680 List of job objects returned.

1681

1682 **Returns**

1683 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1684 value is returned.

1685 **Example**

1686

```

1687           #include "papi.h"
1688
1689           int i;
1690           papi_status_t status;
1691           papi_service_t handle = NULL;
1692           const char* printer_name = "my-printer";
1693           papi_job_t* jobs = NULL;
1694           const char* job_attrs[] =
1695           {
1696            "job-id",
1697            "job-name",
1698            "job-originating-user-name",
1699            "job-state",
1700            "job-state-reasons",

```

```

1701         NULL
1702     };
1703     ...
1704     status = papiServiceCreate(&handle,
1705                               NULL,
1706                               NULL,
1707                               NULL,
1708                               NULL,
1709                               PAPI_ENCRYPT_NEVER,
1710                               NULL);
1711
1712     if (status != PAPI_OK)
1713     {
1714         /* handle the error */
1715         ...
1716     }
1717
1718     status = papiPrinterListJobs(handle,
1719                                 printer_name,
1720                                 job_attrs,
1721                                 PAPI_LIST_JOBS_ALL,
1722                                 0,
1723                                 &jobs);
1724
1725     if (status != PAPI_OK)
1726     {
1727         /* handle the error */
1728         fprintf(stderr, "papiPrinterListJobs failed: %s\n",
1729                papiServiceGetStatusMessage(handle));
1730         ...
1731     }
1732
1733     if (jobs != NULL)
1734     {
1735         for(i=0; jobs[i] != NULL; i++)
1736         {
1737             /* process the job */
1738             ...
1739         }
1740         papiJobListFree(jobs);
1741     }
1742     papiServiceDestroy(handle);

```

1743

1744

See Also

1745

papiJobQuery, papiJobListFree

1746

5.9. papiPrinterGetAttributeList

1747

Description

1748

Get the attribute list associated with a printer object.

1749

Syntax

1750

1751

```

papi_attribute_t** papiPrinterGetAttributeList(
1752     papi_printer_t printer );
1753

```

1754

1755

Inputs

1756

1757

printer

1758

Handle of the printer object.

1759

1760

Outputs

1761

none

1762

Returns

1763

Pointer to the attribute list associated with the printer object.

1764

Example

1765

1766

```

#include "papi.h"

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

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

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

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

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

papiServiceDestroy(handle);

```

1767

1768

1769

1770

1771

1772

1773

1774

1775

1776

1777

1778

1779

1780

1781

1782

1783

1784

1785

1786

1787

1788

1789

1790

1791

1792

1793

1794

1795

1796

1797

1798

1799

1800

1801

1802

1803

1804

1805

1806

1807

1808

1809

1810

See Also

1811

papiPrintersList, papiPrinterQuery

1812

5.10. papiPrinterFree

1813

Description

1814

Free a printer object.

1815

Syntax

1816

1817

```

void papiPrinterFree(
    papi_printer_t printer );

```

1818

1819

1820

1821

Inputs

1822

1823 printer

1824 Handle of the printer object to free.

1825

1826 **Outputs**

1827 none

1828 **Returns**

1829 none

1830 **Example**

1831

```

1832 #include "papi.h"
1833
1834 papi_status_t status;
1835 papi_service_t handle = NULL;
1836 const char* printer_name = "my-printer";
1837 papi_printer_t printer = NULL;
1838 ...
1839 status = papiServiceCreate(&handle,
1840                          NULL,
1841                          NULL,
1842                          NULL,
1843                          NULL,
1844                          PAPI_ENCRYPT_NEVER,
1845                          NULL);
1846
1847 if (status != PAPI_OK)
1848 {
1849     /* handle the error */
1850     ...
1851 }
1852
1853 status = papiPrinterQuery(handle,
1854                          printer_name,
1855                          NULL,
1856                          &printer);
1857
1858 if (status != PAPI_OK)
1859 {
1860     /* handle the error */
1861     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1862            papiServiceGetStatusMessage(handle));
1863     ...
1864 }
1865
1866 if (printer != NULL)
1867 {
1868     /* process the printer object */
1869     ...
1870     papiPrinterFree(printer);
1871 }
1872 papiServiceDestroy(handle);

```

1873

1874 **See Also**

1875 papiPrinterQuery

1876 **5.11. papiPrinterListFree**

1877 **Description**

1878 Free a list of printer objects.

1879 **Syntax**

1880

```

1881 void papiPrinterListFree (
1882     papi_printer_t*   printers );

```

1883

1884

1885 **Inputs**

1886

1887 printers

1888 Pointer to the printer object list to free.

1889

1890 **Outputs**

1891 none

1892 **Returns**

1893 none

1894 **Example**

1895

```

1896 #include "papi.h"
1897
1898 papi_status_t status;
1899 papi_service_t handle = NULL;
1900 const char* printer_name = "my-printer";
1901 papi_printer_t* printers = NULL;
1902 ...
1903 status = papiServiceCreate(&handle,
1904                          NULL,
1905                          NULL,
1906                          NULL,
1907                          NULL,
1908                          PAPI_ENCRYPT_NEVER,
1909                          NULL);
1910
1911 if (status != PAPI_OK)
1912 {
1913     /* handle the error */
1914     ...
1915 }
1916
1917 status = papiPrinterList(handle,
1918                          NULL,
1919                          NULL,
1920                          &printers);
1921
1922 if (status != PAPI_OK)
1923 {
1924     /* handle the error */
1925     fprintf(stderr, "papiPrinterList failed: %s\n",
1926            papiServiceGetStatusMessage(handle));
1927     ...
1928 }
1929
1930 if (printers != NULL)
1931 {
1932     /* process the printer objects */
1933     ...
1934     papiPrinterListFree(printers);
1935 }
1936
1937 papiServiceDestroy(handle);

```

1937

1938 **See Also**

1939 papiPrinterList

1940 Chapter 6. Attributes API

1941 6.1. papiAttributeListAdd

1942 Description

1943 Add an attribute/value to an attribute list. Depending on the `add_flags`, this may
1944 also be used to add values to an existing multivalued attribute. Memory is allocated
1945 and copies of the input arguments are created. It is the caller's responsibility to call
1946 `papiAttributeListFree` when done with the attribute list.

1947 This function is equivalent to the `papiAttributeListAddString`,
1948 `papiAttributeListAddInteger`, etc. functions defined later in this chapter.

1949 Syntax

1950

```
1951 papi_status_t papiAttributeListAdd(  
1952     papi_attribute_t*** attrs,  
1953     const int add_flags,  
1954     const char* name,  
1955     const papi_attribute_value_type_t type,  
1956     const papi_attribite_value_t* value );  
1957
```

1958

1959 Inputs

1960

1961 `attrs`

1962 Points to an attribute list. If a NULL value is passed, this function will allocate
1963 the attribute list.

1964 `add_flags`

1965 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
1966 that indicates how to handle the request.

1967 `name`

1968 Points to the name of the attribute to add.

1969 `type`

1970 The type of values for this attribute.

1971 `value`

1972 Points to the attribute value to be added.

1973

1974 Outputs

1975

1976 `attrs`

1977 The attribute list is updated.

1978

1979

Returns

1980

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

1981

1982

Example

1983

1984

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

1985

1986

1987

1988

1989

1990

1991

1992

1993

1994

1995

1996

1997

See Also

1998

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

1999

2000

2001

6.2. papiAttributeListAddString

2002

Description

2003

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.

2004

2005

2006

2007

Syntax

2008

2009

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

2010

2011

2012

2013

2014

2015

2016

Inputs

2017

2018 `attrs`

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

2019

2020

2021 `add_flags`

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

2022

2023

2024 name
 2025 Points to the name of the attribute to add.

2026 value
 2027 The value to be added.

2028

2029 **Outputs**

2030

2031 attrs
 2032 The attribute list is updated.

2033

2034 **Returns**

2035 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2036 value is returned.

2037 **Example**

2038

```
2039 #include "papi.h"
2040
2041 papi_attribute_t** attrs = NULL;
2042 ...
2043 papiAttributeListAddString(&attrs,
2044                           PAPI_EXCL,
2045                           "job-name",
2046                           "My job" );
2047 ...
2048 papiAttributeListFree(attrs);
2049
```

2050

2051 **See Also**

2052 papiAttributeListFree, papiAttributeListAdd

2053 **6.3. papiAttributeListAddInteger**

2054 **Description**

2055 Add an integer-valued attribute to an attribute list. Depending on the add_flags,
 2056 this may also be used to add values to an existing multivalued attribute. Memory is
 2057 allocated and copies of the input arguments are created. It is the caller's
 2058 responsibility to call papiAttributeListFree when done with the attribute list.

2059 **Syntax**

2060

```
2061 papi_status_t papiAttributeListAddInteger(
2062     papi_attribute_t*** attrs,
2063     const int add_flags,
2064     const char* name,
2065     const int value );
2066
```

2067

2068 **Inputs**

2069

2070 attrs

2071 Points to an attribute list. If a NULL value is passed, this function will allocate
2072 the attribute list.

2073 add_flags

2074 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
2075 that indicates how to handle the request.

2076 name

2077 Points to the name of the attribute to add.

2078 value

2079 The value to be added.

2080

2081 **Outputs**

2082

2083 attrs

2084 The attribute list is updated.

2085

2086 **Returns**

2087 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2088 value is returned.

2089 **Example**

2090

```
2091           #include "papi.h"
2092
2093           papi_attribute_t** attrs = NULL;
2094           ...
2095           papiAttributeListAddInteger(&attrs,
2096                                       PAPI_EXCL,
2097                                       "copies",
2098                                       3 );
2099           ...
2100           papiAttributeListFree(attrs);
2101
```

2102

2103 **See Also**

2104 papiAttributeListFree, papiAttributeListAdd

2105 **6.4. papiAttributeListAddBoolean**2106 **Description**

2107 Add a boolean-valued attribute to an attribute list. Depending on the add_flags,
2108 this may also be used to add values to an existing multivalued attribute. Memory is
2109 allocated and copies of the input arguments are created. It is the caller's
2110 responsibility to call papiAttributeListFree when done with the attribute list.

2111 **Syntax**

2112

```

2113     papi_status_t papiAttributeListAddBoolean(
2114         papi_attribute_t*** attrs,
2115         const int add_flags,
2116         const char* name,
2117         const char value );
2118 
```

2119

2120 **Inputs**

2121

2122 attrs

2123 Points to an attribute list. If a NULL value is passed, this function will allocate
 2124 the attribute list.

2125 add_flags

2126 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2127 that indicates how to handle the request.

2128 name

2129 Points to the name of the attribute to add.

2130 value

2131 The value (PAPI_FALSE or PAPI_TRUE) to be added.

2132

2133 **Outputs**

2134

2135 attrs

2136 The attribute list is updated.

2137

2138 **Returns**

2139 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2140 value is returned.

2141 **Example**

2142

```

2143     #include "papi.h"
2144
2145     papi_attribute_t** attrs = NULL;
2146     ...
2147     papiAttributeListAddBoolean(&attrs,
2148                               PAPI_EXCL,
2149                               "color-supported",
2150                               PAPI_TRUE );
2151     ...
2152     papiAttributeListFree(attrs);
2153 
```

2154

2155 **See Also**
 2156 papiAttributeListFree, papiAttributeListAdd

2157 **6.5. papiAttributeListAddRange**

2158 **Description**

2159 Add a range-valued attribute to an attribute list. Depending on the `add_flags`, this
 2160 may also be used to add values to an existing multivalued attribute. Memory is
 2161 allocated and copies of the input arguments are created. It is the caller's
 2162 responsibility to call `papiAttributeListFree` when done with the attribute list.

2163 **Syntax**

2164

```
2165 papi_status_t papiAttributeListAddRange(  
2166     papi_attribute_t*** attrs,  
2167     const int add_flags,  
2168     const char* name,  
2169     const int lower,  
2170     const int upper );  
2171
```

2172

2173 **Inputs**

2174

2175 `attrs`

2176 Points to an attribute list. If a NULL value is passed, this function will allocate
 2177 the attribute list.

2178 `add_flags`

2179 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 2180 that indicates how to handle the request.

2181 `name`

2182 Points to the name of the attribute to add.

2183 `lower`

2184 The lower range value. This value must be less than or equal to the upper
 2185 range value.

2186 `upper`

2187 The upper range value. This value must be greater than or equal to the lower
 2188 range value.

2189

2190 **Outputs**

2191

2192 `attrs`

2193 The attribute list is updated.

2194

2195

Returns

2196

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

2197

2198

Example

2199

2200

```
#include "papi.h"
2201
2202 papi_attribute_t** attrs = NULL;
2203 ...
2204 papiAttributeListAddRange (&attrs,
2205                             PAPI_EXCL,
2206                             "job-k-octets-supported",
2207                             1,
2208                             100000 );
2209 ...
2210 papiAttributeListFree (attrs);
2211
```

2212

2213

See Also

2214

papiAttributeListFree

2215

6.6. papiAttributeListAddResolution

2216

Description

2217

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.

2218

2219

2220

2221

Syntax

2222

2223

```
papi_status_t papiAttributeListAddResolution(
2224     papi_attribute_t*** attrs,
2225     const int add_flags,
2226     const char* name,
2227     const papi_res_t units,
2228     const int xres,
2229     const int yres );
2230
```

2231

2232

Inputs

2233

2234

attrs

2235

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

2236

2237

add_flags

2238

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

2239

2240 name
2241 Points to the name of the attribute to add.
2242 units
2243 The units of the resolution values provided.
2244 xres
2245 The X-axis resolution value.
2246 yres
2247 The Y-axis resolution value.
2248

2249 **Outputs**

2250
2251 attrs
2252 The attribute list is updated.
2253

2254 **Returns**

2255 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2256 value is returned.

2257 **Example**

2258
2259

```
#include "papi.h"  
2260  
2261                   papi_attribute_t** attrs = NULL;  
2262                   ...  
2263                   papiAttributeListAddResolution(&attrs,  
2264                                                   PAPI_EXCL,  
2265                                                   "printer-resolution",  
2266                                                   PAPI_RES_PER_INCH,  
2267                                                   300,  
2268                                                   300 );  
2269                   ...  
2270                   papiAttributeListFree(attrs);  
2271
```

2272
2273 **See Also**

2274 papiAttributeListFree

2275 **6.7. papiAttributeListAddDatetime**

2276 **Description**

2277 Add a date/time-valued attribute to an attribute list. Depending on the add_flags,
2278 this may also be used to add values to an existing multivalued attribute. Memory is
2279 allocated and copies of the input arguments are created. It is the caller's
2280 responsibility to call papiAttributeListFree when done with the attribute list.

2281 **Syntax**

2282


```

2283     papi_status_t papiAttributeListAddDatetime(
2284         papi_attribute_t*** attrs,
2285         const int add_flags,
2286         const char* name,
2287         const time_t date_time );
2288

```

2289

2290 **Inputs**

2291

2292 attrs

2293 Points to an attribute list. If a NULL value is passed, this function will allocate
 2294 the attribute list.

2295 add_flags

2296 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
 2297 that indicates how to handle the request.

2298 name

2299 Points to the name of the attribute to add.

2300 date_time

2301 The date/time value.

2302

2303 **Outputs**

2304

2305 attrs

2306 The attribute list is updated.

2307

2308 **Returns**

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

2311 **Example**

2312

```

2313 #include "papi.h"
2314
2315 papi_attribute_t** attrs = NULL;
2316 time_t date_time
2317 ...
2318 time(&date_time);
2319 papiAttributeListAddDatetime(&attrs,
2320                             PAPI_EXCL,
2321                             "date-time-at-creation",
2322                             date_time );
2323
2324 ...
2325 papiAttributeListFree(attrs);

```

2326

2327 **See Also**

2328 papiAttributeListFree

2329 **6.8. papiAttributeDelete**

2330 **Description**

2331 Delete an attribute from an attribute list.

2332 **Syntax**

2333

```
2334           papi_status_t papiAttributeDelete(  
2335                           papi_attribute_t*** attrs,  
2336                           const char* name);  
2337
```

2338

2339 **Inputs**

2340

2341 attrs

2342 Points to an attribute list.

2343 name

2344 Points to the name of the attribute to delete.

2345

2346 **Outputs**

2347

2348 attrs

2349 The attribute list is updated.

2350

2351 **Returns**

2352 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2353 value is returned.

2354 **Example**

2355

```
2356           #include "papi.h"  
2357           papi_attribute_t** attrs = NULL;  
2358           ...  
2359           papiAttributeDelete(&attrs,  
2360                               "copies" );  
2361           ...  
2362           ...  
2363
```

2364

2365 **See Also**

2366 papiAttributeListFree

2367 6.9. papiAttributeListGetValue

2368 Description

2369 Get an attribute's value from an attribute list.

2370 This function is equivalent to the papiAttributeListGetString,
2371 papiAttributeListGetInteger, etc. functions defined later in this chapter.

2372 Syntax

2373

```
2374 papi_status_t papiAttributeListGetValue(
2375     papi_attribute_t*** attrs,
2376     void** iterator,
2377     const char* name,
2378     const papi_attribute_value_type_t type,
2379     papi_attribute_value_t* value );
2380
```

2381

2382 Inputs

2383

2384 attrs

2385 Points to an attribute list.

2386 iterator

2387 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
2388 then only the first value is returned, even if the attribute is multivalued. If the
2389 argument points to a void* that is set to NULL, then the first attribute value is
2390 returned and the iterator can then be passed in unchanged on subsequent calls
2391 to this function to get the remaining values.

2392 name

2393 Points to the name of the attribute whose value to get.

2394 type

2395 The type of values for this attribute.

2396

2397 Outputs

2398

2399 value

2400 Points to the attribute value to be returned.

2401

2402 Returns

2403 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2404 value is returned.

2405

Example

2406

2407

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

2408

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

2409

```
char* job_name_value = NULL;
```

2410

```
...
```

2411

```
papiAttributeListGetValue(&attrs,
```

2412

```
NULL,
```

2413

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

2414

```
PAPI_STRING,
```

2415

```
&job_name_value );
```

2416

```
if (job_name_value != NULL)
```

2417

```
{
```

2418

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

2419

```
    ...
```

2420

```
}
```

2421

```
...  
papiAttributeListFree(attrs);
```

2422

See Also

2423

papiAttributeListFree, papiAttributeListGetString, papiAttributeListGetInteger,

2424

papiAttributeListGetBoolean, papiAttributeListGetRange,

2425

papiAttributeListGetResolution, papiAttributeListGetDatetime

2426

6.10. papiAttributeListGetString

2427

Description

2428

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

2429

Syntax

2430

2431

```
papi_status_t papiAttributeListGetString(  
2432     papi_attribute_t*** attrs,  
2433     void** iterator,  
2434     const char* name,  
2435     char** value );
```

2436

2437

2438

2439

2440

2441

2442

Inputs

2443

2444 attrs

Points to an attribute list.

2446 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.

2452 name

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

2453

2454

2455

Outputs

2456

2457 value

2458

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

2459

2460

Returns

2461

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

2462

2463

Example

2464

2465

```
#include "papi.h"

papi_attribute_t** attrs = NULL;
char* job_name_value = NULL;
...
papiAttributeListGetString(&attrs,
    NULL,
    "job-name",
    &job_name_value );
if (job_name_value != NULL)
{
    /* process the value */
    ...
}
...
papiAttributeListFree(attrs);
```

2466

2467

2468

2469

2470

2471

2472

2473

2474

2475

2476

2477

2478

2479

2480

2481

2482

2483

See Also

2484

papiAttributeListFree, papiAttributeListGetValue

2485

6.11. papiAttributeListGetInteger

2486

Description

2487

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

2488

Syntax

2489

2490

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

2491

2492

2493

2494

2495

2496

2497

Inputs

2498

2499

attrs

2500

Points to an attribute list.

2501 iterator

2502 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2503 then only the first value is returned, even if the attribute is multivalued. If the
 2504 argument points to a void* that is set to NULL, then the first attribute value is
 2505 returned and the iterator can then be passed in unchanged on subsequent calls
 2506 to this function to get the remaining values.

2507 name

2508 Points to the name of the attribute whose value to get.
 2509

2510 **Outputs**

2511

2512 value

2513 Pointer to the int where the value is returned.
 2514

2515 **Returns**

2516 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2517 value is returned.

2518 **Example**

2519

```

2520 #include "papi.h"
2521
2522 papi_attribute_t** attrs = NULL;
2523 int copies = 0;
2524 ...
2525 papiAttributeListGetInteger(&attrs,
2526                             NULL,
2527                             "copies",
2528                             &copies );
2529 /* process the value */
2530 ...
2531 papiAttributeListFree(attrs);
2532

```

2533

2534 **See Also**

2535 papiAttributeListFree, papiAttributeListGetValue

2536 6.12. papiAttributeListGetBoolean

2537 **Description**

2538 Get an boolean-valued attribute's value from an attribute list.

2539 **Syntax**

2540

```

2541 papi_status_t papiAttributeListGetBoolean(
2542     papi_attribute_t*** attrs,
2543     void** iterator,
2544     const char* name,
2545     char* value );
2546

```

2547

2548

Inputs

2549

2550 attrs

2551

Points to an attribute list.

2552 iterator

2553

(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.

2554

2555

2556

2557

2558 name

2559

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

2560

2561

Outputs

2562

2563 value

2564

Pointer to the char where the value is returned.

2565

2566

Returns

2567

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

2568

2569

Example

2570

2571

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

2572

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

2573

```
char color_supp = PAPI_FALSE;
```

2574

```
...
```

2575

```
papiAttributeListGetBoolean(&attrs,
```

2576

```
NULL,
```

2577

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

2578

```
&color_supp );
```

2579

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

2580

```
...
```

2581

```
papiAttributeListFree(attrs);
```

2582

2583

2584

See Also

2585

papiAttributeListFree, papiAttributeListGetValue

2586

2587

6.13. papiAttributeListGetRange

2588

Description

2589

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

2590 **Syntax**

2591

```

2592     papi_status_t papiAttributeListGetRange(
2593         papi_attribute_t*** attrs,
2594         void** iterator,
2595         const char* name,
2596         int* lower,
2597         int* upper );
2598

```

2599

2600 **Inputs**

2601

2602 attrs

2603 Points to an attribute list.

2604 iterator

2605 (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.

2610 name

2611 Points to the name of the attribute whose value to get.

2612

2613 **Outputs**

2614

2615 lower

2616 Pointer to the int where the lower range value is returned.

2617 upper

2618 Pointer to the int where the upper range value is returned.

2619

2620 **Returns**

2621 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is returned.

2623 **Example**

2624

```

2625     #include "papi.h"
2626
2627     papi_attribute_t** attrs = NULL;
2628     int lower = 0;
2629     int upper = 0;
2630     ...
2631     papiAttributeListGetRange(&attrs,
2632                             NULL,
2633                             "job-k-octets-supported",
2634                             &lower,

```



```

2635         &upper );
2636     /* process the value */
2637     ...
2638     papiAttributeListFree(attrs);
2639

```

2640

2641 **See Also**

2642 papiAttributeListFree, papiAttributeListGetValue

2643 **6.14. papiAttributeListGetResolution**2644 **Description**

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

2646 **Syntax**

2647

```

2648 papi_status_t papiAttributeListGetResolution(
2649     papi_attribute_t*** attrs,
2650     void** iterator,
2651     const char* name,
2652     int* xres,
2653     int* yres,
2654     papi_res_t* units );
2655

```

2656

2657 **Inputs**

2658

2659 attrs

2660 Points to an attribute list.

2661 iterator

2662 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2663 then only the first value is returned, even if the attribute is multivalued. If the
 2664 argument points to a void* that is set to NULL, then the first attribute value is
 2665 returned and the iterator can then be passed in unchanged on subsequent calls
 2666 to this function to get the remaining values.

2667 name

2668 Points to the name of the attribute whose value to get.

2669

2670 **Outputs**

2671

2672 xres

2673 Pointer to the int where the X-resolution value is returned.

2674 yres

2675 Pointer to the int where the Y-resolution value is returned.

2676 units

2677 Pointer to the variable where the resolution-units value is returned.

2678

2679 **Returns**2680 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2681 value is returned.2682 **Example**

2683

```

2684 #include "papi.h"
2685
2686 papi_attribute_t** attrs = NULL;
2687 int xres = 0;
2688 int yres = 0;
2689 papi_res_t units;
2690 ...
2691 papiAttributeListGetResolution(&attrs,
2692                               NULL,
2693                               "printer-resolution",
2694                               &xres,
2695                               &yres,
2696                               &units );
2697 /* process the value */
2698 ...
2699 papiAttributeListFree(attrs);
2700

```

2701

2702 **See Also**

2703 papiAttributeListFree, papiAttributeListGetValue

2704 **6.15. papiAttributeListGetDatetime**2705 **Description**

2706 Get a date/time-valued attribute's value from an attribute list.

2707 **Syntax**

2708

```

2709 papi_status_t papiAttributeListGetDatetime(
2710     papi_attribute_t*** attrs,
2711     void** iterator,
2712     const char* name,
2713     time_t* date_time );
2714

```

2715

2716 **Inputs**

2717

2718 attrs

2719 Points to an attribute list.

2720 iterator

2721 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL
 2722 then only the first value is returned, even if the attribute is multivalued. If the
 2723 argument points to a void* that is set to NULL, then the first attribute value is

2724 returned and the iterator can then be passed in unchanged on subsequent calls
 2725 to this function to get the remaining values.

2726 name

2727 Points to the name of the attribute whose value to get.

2728

2729 **Outputs**

2730

2731 date_time

2732 Pointer to the variable where the date/time value is returned.

2733

2734 **Returns**

2735 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2736 value is returned.

2737 **Example**

2738

```

2739 #include "papi.h"
2740
2741 papi_attribute_t** attrs = NULL;
2742 time_t date_time;
2743 ...
2744 papiAttributeListGetDatetime(&attrs,
2745                             NULL,
2746                             "date-time-at-creation",
2747                             &date_time );
2748 /* process the value */
2749 ...
2750 papiAttributeListFree(attrs);
2751
```

2752

2753 **See Also**

2754 papiAttributeListFree, papiAttributeListGetValue

2755 **6.16. papiAttributeListFree**

2756 **Description**

2757 Frees an attribute list.

2758 **Syntax**

2759

```

2760 void papiAttributeListFree(
2761     const papi_attribute_t** attrs );
2762
```

2763

2764 **Inputs**

2765

2766 attrs
2767 Attribute list to be freed.

2768

2769 **Outputs**

2770 none

2771 **Returns**

2772 none

2773 **Example**

2774

```
2775 #include "papi.h"  
2776  
2777 papi_attribute_t** attrs = NULL;  
2778 ...  
2779 papiAttributeListAddString(&attrs,  
2780                             "job-name",  
2781                             PAPI_EXCL,  
2782                             1,  
2783                             "My job" );  
2784 ...  
2785 papiAttributeListFree(attrs);  
2786
```

2787

2788 **See Also**

2789 papiAttributeListAddString, etc.

2790 **6.17. papiAttributeListFind**

2791 **Description**

2792 Find an attribute in an attribute list.

2793 **Syntax**

2794

```
2795 papi_attribute_t* papiAttributeListFind(  
2796     const papi_attribute_t** attrs,  
2797     const char* name );  
2798
```

2799

2800 **Inputs**

2801

2802 attrs
2803 Attribute list to be searched.

2804 name
2805 Pointer to the name of the attribute to find.

2806

2807 **Outputs**

2808 none

2809

Returns

2810

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

2811

2812

Example

2813

2814

```
#include "papi.h"

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

2815

2816

2817

2818

2819

2820

2821

2822

2823

2824

2825

2826

2827

2828

2829

2830

See Also

2831

papiAttributeListGetNext

2832

6.18. papiAttributeListGetNext

2833

Description

2834

Get the next attribute in an attribute list.

2835

Syntax

2836

2837

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

2838

2839

2840

2841

2842

Inputs

2843

2844

attrs

Attribute list to be used.

2845

2846

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.

2847

2848

2849

2850

Outputs

2851

none

2852

Returns

2853

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

2854

2855

Example

2856

2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874

```
#include "papi.h"

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

2875

2876

See Also

2877

papiAttributeListFind

2878 Chapter 7. Job API

2879 7.1. papiJobSubmit

2880 Description

2881 Submits a print job having the specified attributes to the specified printer.

2882 Syntax

2883

```
2884 papi_status_t papiJobSubmit(  
2885     papi_service_t    handle,  
2886     const char*       printer_name,  
2887     const papi_attribute_t** job_attributes,  
2888     const papi_job_ticket_t* job_ticket,  
2889     const char**      file_names,  
2890     papi_job_t*       job );  
2891
```

2892

2893 Inputs

2894

2895 handle

2896 Handle to the print service to use.

2897 printer_name

2898 Pointer to the name of the printer to which the job is to be submitted.

2899 job_attributes

2900 (optional) The list of attributes describing the job and how it is to be printed. If
2901 options are specified here and also in the job ticket data, the value specified
2902 here takes precedence. If this is NULL then only default attributes and
2903 (optionally) a job ticket is submitted with the job.

2904 job_ticket

2905 (optional) Pointer to structure specifying the job ticket. If this argument is
2906 NULL, then no job ticket is used with the job.

2907 file_names

2908 NULL terminated list of pointers to names of files to print.

2909

2910 Outputs

2911

2912 job

2913 The resulting job object representing the submitted job.

2914

2915

Returns

2916

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

2917

2918

Example

2919

2920

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

2921

```
papi_status_t status;
```

2922

```
papi_service_t handle = NULL;
```

2923

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

2924

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

2925

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

2926

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

2927

```
papi_job_t job = NULL;
```

2928

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

2929

```
        PAPI_ENCRYPT_IF_REQUESTED, NULL);
```

2930

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

2931

```
{
```

2932

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

2933

```
    ...
```

2934

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

2935

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

2936

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

2937

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

2938

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

2939

```
        printer,
```

2940

```
        attrs,
```

2941

```
        ticket,
```

2942

```
        files,
```

2943

```
        &job);
```

2944

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

2945

```
{
```

2946

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

2947

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

2948

```
    ...
```

2949

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

2950

```
{
```

2951

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

2952

```
    papiJobFree(job);
```

2953

```
}
```

2954

```
papiServiceDestroy(handle);
```

2955

2956

2957

Description

2958

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.

2959

2960

2961

Syntax

2962

2963

2964

```
papi_status_t papiJobValidate(
```

2965

```
        papi_service_t
```

```
        handle,
```

2966

```
        const char*
```

```
        printer_name,
```

2967

```
        const papi_attribute_t**
```

```
        job_attributes,
```

2968


```

2979         const papi_job_ticket_t*   job_ticket,
2980         const char**               file_names,
2981         papi_job_t*                job );
2982

```

2983

2984 **Inputs**

2985

2986 handle

2987 Handle to the print service to use.

2988 printer_name

2989 Pointer to the name of the printer against which the job is to be validated.

2990 job_attributes

2991 (optional) The list of attributes describing the job and how it is to be printed. If
 2992 options are specified here and also in the job ticket data, the value specified
 2993 here takes precedence. If this is NULL then only default attributes and
 2994 (optionally) a job ticket is submitted with the job.

2995 job_ticket

2996 (optional) Pointer to structure specifying the JDF job ticket. If this argument is
 2997 NULL, then no job ticket is used with the job.

2998 file_names

2999 NULL terminated list of pointers to names of files to validate.

3000

3001 **Outputs**

3002

3003 job

3004 The resulting job object representing what would be the submitted job.

3005

3006 **Returns**

3007 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3008 value is returned.

3009 **Example**

3010

```

3011 #include "papi.h"
3012
3013 papi_status_t status;
3014 papi_service_t handle = NULL;
3015 const char* printer = "my-printer";
3016 const papi_attribute_t** attrs = NULL;
3017 const papi_job_ticket_t* ticket = NULL;
3018 const char* files[] = { "/etc/motd", NULL };
3019 papi_job_t job = NULL;
3020
3021 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
3022                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
3023 if (status != PAPI_OK)
3024 {

```

```

3025         /* handle the error */
3026         ...
3027     }
3028
3029     papiAttributeListAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
3030                               PAPI_STRING, 1, "test job");
3031     papiAttributeListAddInteger(&attrs, "Copies", PAPI_ATTR_EXCL,
3032                                PAPI_INTEGER, 1, 4);
3033
3034     status = papiJobValidate(handle,
3035                              printer,
3036                              attrs,
3037                              ticket,
3038                              files,
3039                              &job);
3040
3041     if (status != PAPI_OK)
3042     {
3043         fprintf(stderr, "papiJobValidate failed: %s\n",
3044                papiStatusString(status));
3045         ...
3046     }
3047
3048     if (job != NULL)
3049     {
3050         ...
3051         papiJobFree(job);
3052     }
3053
3054     papiServiceDestroy(handle);

```

3055

3056

See Also

3057

papiJobSubmit, papiJobFree

3058

7.3. papiJobQuery

3059

Description

3060

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

3061

Syntax

3062

```

3063     papi_status_t papiJobQuery(
3064         papi_service_t    handle,
3065         const char*       printer_name,
3066         const int32_t     job_id,
3067         const char*       requested_attrs[],
3068         papi_job_t*      job );
3069

```

3070

3071

Inputs

3072

3073 handle

3074

Handle to the print service to use.

3075 printer_name

3076

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

3077 job_id

3078

The ID number of the job to be queried.

3079 requested_attrs

3080 NULL terminated array of attributes to be queried. If NULL is passed then all
 3081 available attributes are queried. (NOTE: The job may return more attributes
 3082 than you requested. This is merely an advisory request that may reduce the
 3083 amount of data returned if the printer/server supports it.)

3084

3085 **Outputs**

3086

3087 job

3088 The returned job object containing the requested attributes.

3089

3090 **Returns**

3091 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3092 value is returned.

3093 **Example**

3094

```

3095 #include "papi.h"
3096
3097 papi_status_t status;
3098 papi_service_t handle = NULL;
3099 const char* printer_name = "my-printer";
3100 papi_job_t job = NULL;
3101 int32_t job_id = 12;
3102 const char* job_attrs[] =
3103 {
3104     "job-id",
3105     "job-name",
3106     "job-originating-user-name",
3107     "job-state",
3108     "job-state-reasons",
3109     NULL
3110 };
3111
3112 ...
3113 status = papiServiceCreate(&handle,
3114                             NULL,
3115                             NULL,
3116                             NULL,
3117                             NULL,
3118                             PAPI_ENCRYPT_NEVER,
3119                             NULL);
3120
3121 if (status != PAPI_OK)
3122 {
3123     /* handle the error */
3124     ...
3125 }
3126
3127 status = papiJobQuery(handle,
3128                       printer_name,
3129                       job_id,
3130                       job_attrs,
3131                       &job);
3132
3133 if (status != PAPI_OK)
3134 {
3135     /* handle the error */
3136     fprintf(stderr, "papiJobQuery failed: %s\n",
3137             papiServiceGetStatusMessage(handle));
3138     ...
3139 }
3140
3141 if (job != NULL)
3142 {
3143     /* process the job */
3144     ...
3145     papiJobFree(job);
3146 }
3147
3148 papiServiceDestroy(handle);
  
```

3147

3148

See Also

3149

papiJobFree, papiPrinterListJobs, papiJobModify

3150

7.4. papiJobModify

3151

Description

3152

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

3153

Syntax

3154

3155

```
papi_status_t papiJobModify(  
    papi_service_t    handle,  
    const char*       printer_name,  
    const int32_t     job_id,  
    const papi_attribute_t** attrs,  
    papi_job_t*      job );
```

3156

3157

3158

3159

3160

3161

3162

3163

Inputs

3164

3165

handle

3166

Handle to the print service to use.

3167

printer_name

3168

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

3169

job_id

3170

The ID number of the job to be modified.

3171

attrs

3172

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

3173

3174

Outputs

3175

3176

job

3177

The modified job object.

3178

3179

Returns

3180

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

3181

value is returned.

3182

Example

3183

3184

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

```

3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232

```

```

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

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

papiAttributeListAddInteger(&attrs,
                             PAPI_EXCL,
                             "copies",
                             3);

status = papiJobModify(handle,
                       printer_name,
                       job_id,
                       attrs,
                       &job);

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

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

papiServiceDestroy(handle);

```

3233

3234

See Also

3235

papiJobQuery, papiJobFree, papiPrinterListJobs

3236

7.5. papiJobCancel

3237

Description

3238

Cancel the specified print job.

3239

Syntax

3240

3241

3242

3243

3244

3245

```

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

```

3246

3247

Inputs

3248

3249 handle

3250 Handle to the print service to use.

3251 printer_name

3252 Pointer to the name or URI of the printer to which the job was submitted.

3253 job_id

3254 The ID number of the job to be cancelled.

3255

3256 **Outputs**

3257 none

3258 **Returns**

3259 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3260 value is returned.

3261 **Example**

3262

```

3263 #include "papi.h"
3264
3265 papi_status_t status;
3266 papi_service_t handle = NULL;
3267 const char* printer_name = "my-printer";
3268 int32_t job_id = 12;
3269 ...
3270 status = papiServiceCreate(&handle,
3271                          NULL,
3272                          NULL,
3273                          NULL,
3274                          NULL,
3275                          PAPI_ENCRYPT_NEVER,
3276                          NULL);
3277
3278 if (status != PAPI_OK)
3279 {
3280     /* handle the error */
3281     ...
3282 }
3283
3284 status = papiJobCancel(handle,
3285                       printer_name,
3286                       job_id);
3287
3288 if (status != PAPI_OK)
3289 {
3290     /* handle the error */
3291     fprintf(stderr, "papiJobCancel failed: %s\n",
3292           papiServiceGetStatusMessage(handle));
3293     ...
3294 }
3295 papiServiceDestroy(handle);

```

3296

3297 **See Also**

3298 papiPrinterListJobs, papiPrinterPurgeJobs

3299 7.6. papiJobHold

3300 **Description**

3301 Holds the specified print job and prevents it from being scheduled for printing.
3302 This operation is optional and may not be supported by all printers/servers. Use
3303 papiJobRelease to undo the effects of this operation, or specify the hold_until
3304 argument to automatically release the job at a specific time.

3305 **Syntax**

3306

```

3307     papi_status_t papiJobHold(
3308         papi_service_t     handle,
3309         const char*         printer_name,
3310         const int32_t       job_id,
3311         const char*         hold_until,
3312         const time_t*       hold_until_time );
3313

```

3314

3315 **Inputs**

3316

3317 handle

3318 Handle to the print service to use.

3319 printer_name

3320 Pointer to the name or URI of the printer to which the job was submitted.

3321 job_id

3322 The ID number of the job to be held.

3323 hold_until

3324 (optional) Specifies the time when the job will be automatically released for
 3325 printing. If NULL, the job is held until explicitly released by calling
 3326 papiJobRelease. If specified, the value must be one of the strings "indefinite"
 3327 (same effect as passing NULL), "day-time", "evening", "night", "weekend",
 3328 "second-shift", "third-shift", or "timed". For values other than "indefinite" and
 3329 "timed", the printer/server must define exact times associated with these
 3330 values and it may make these associations configurable. If "timed" is specified,
 3331 then the hold_until_time argument is used.

3332 hold_until_time

3333 (optional) Specifies the time when the job will be automatically released for
 3334 printing. This argument is ignored unless "timed" is passed as the hold_until
 3335 argument.

3336

3337 **Outputs**

3338 none

3339 **Returns**

3340 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 3341 value is returned.

3342 **Example**

3343

```

3344 #include "papi.h"
3345
3346 papi_status_t status;
3347 papi_service_t handle = NULL;

```

```

3348     const char* printer_name = "my-printer";
3349     int32_t job_id = 12;
3350     ...
3351     status = papiServiceCreate(&handle,
3352                               NULL,
3353                               NULL,
3354                               NULL,
3355                               NULL,
3356                               PAPI_ENCRYPT_NEVER,
3357                               NULL);
3358
3359     if (status != PAPI_OK)
3360     {
3361         /* handle the error */
3362         ...
3363     }
3364
3365     status = papiJobHold(handle,
3366                         printer_name,
3367                         job_id,
3368                         NULL,
3369                         NULL);
3370
3371     if (status != PAPI_OK)
3372     {
3373         /* handle the error */
3374         fprintf(stderr, "papiJobHold failed: %s\n",
3375                papiServiceGetStatusMessage(handle));
3376         ...
3377     }
3378     papiServiceDestroy(handle);

```

3379

3380

See Also

3381

papiJobRelease

3382

7.7. papiJobRelease

3383

Description

3384

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

3385

3386

3387

Syntax

3388

3389

```

papi_status_t papiJobRelease(
3390     papi_service_t handle,
3391     const char* printer_name,
3392     const int32_t job_id );
3393

```

3394

3395

Inputs

3396

3397 handle

3398

Handle to the print service to use.

3399

printer_name

3400

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

3401

job_id

3402

The ID number of the job to be released.

3403

3404

Outputs

3405

none

3406

Returns

3407

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

3408

3409

Example

3410

```

3411 #include "papi.h"
3412
3413 papi_status_t status;
3414 papi_service_t handle = NULL;
3415 const char* printer_name = "my-printer";
3416 int32_t job_id = 12;
3417 ...
3418 status = papiServiceCreate(&handle,
3419                          NULL,
3420                          NULL,
3421                          NULL,
3422                          NULL,
3423                          PAPI_ENCRYPT_NEVER,
3424                          NULL);
3425
3426 if (status != PAPI_OK)
3427 {
3428     /* handle the error */
3429     ...
3430 }
3431
3432 status = papiJobRelease(handle,
3433                       printer_name,
3434                       job_id);
3435
3436 if (status != PAPI_OK)
3437 {
3438     /* handle the error */
3439     fprintf(stderr, "papiJobRelease failed: %s\n",
3440            papiServiceGetStatusMessage(handle));
3441     ...
3442 }
3443 papiServiceDestroy(handle);

```

3444

3445

See Also

3446

papiJobHold

3447

7.8. papiJobRestart

3448

Description

3449

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.

3450

3451

3452

Syntax

3453

3454

```

3455 papi_status_t papiJobRestart(
3456             papi_service_t handle,
3457             const char* printer_name,
3458             const int32_t job_id );

```

3459

3460 **Inputs**

3461

3462 handle

3463 Handle to the print service to use.

3464 printer_name

3465 Pointer to the name or URI of the printer to which the job was submitted.

3466 job_id

3467 The ID number of the job to be restarted.

3468

3469 **Outputs**

3470 none

3471 **Returns**3472 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3473 value is returned.3474 **Example**

3475

```

3476           #include "papi.h"
3477
3478           papi_status_t status;
3479           papi_service_t handle = NULL;
3480           const char* printer_name = "my-printer";
3481           int32_t job_id = 12;
3482           ...
3483           status = papiServiceCreate(&handle,
3484                                      NULL,
3485                                      NULL,
3486                                      NULL,
3487                                      NULL,
3488                                      PAPI_ENCRYPT_NEVER,
3489                                      NULL);
3490
3491           if (status != PAPI_OK)
3492           {
3493               /* handle the error */
3494               ...
3495           }
3496
3497           status = papiJobRestart(handle,
3498                                   printer_name,
3499                                   job_id);
3500
3501           if (status != PAPI_OK)
3502           {
3503               /* handle the error */
3504               fprintf(stderr, "papiJobRestart failed: %s\n",
3505                        papiServiceGetStatusMessage(handle));
3506               ...
3507           }
3508           papiServiceDestroy(handle);

```

3509

3510 **See Also**

3511 papiPrinterListJobs

3512 **7.9. papiJobGetAttributeList**3513 **Description**

3514 Get the attribute list associated with a job object.

3515

Syntax

3516

```

3517 papi_attribute_t** papiJobGetAttributeList(
3518     papi_job_t     job );
3519

```

3520

3521

Inputs

3522

3523 job

Handle of the job object.

3525

3526

Outputs

3527

none

3528

Returns

3529

Pointer to the attribute list associated with the job object.

3530

Example

3531

```

3532 #include "papi.h"
3533
3534 papi_status_t status;
3535 papi_service_t handle = NULL;
3536 const char* printer_name = "my-printer";
3537 papi_job_t job = NULL;
3538 papi_attribute_list* attrs = NULL;
3539 ...
3540 status = papiServiceCreate(&handle,
3541     NULL,
3542     NULL,
3543     NULL,
3544     NULL,
3545     PAPI_ENCRYPT_NEVER,
3546     NULL);
3547
3548 if (status != PAPI_OK)
3549 {
3550     /* handle the error */
3551     ...
3552 }
3553
3554 status = papiJobQuery(handle,
3555     printer_name,
3556     67,
3557     NULL,
3558     &job);
3559
3560 if (status != PAPI_OK)
3561 {
3562     /* handle the error */
3563     fprintf(stderr, "papiJobQuery failed: %s\n",
3564         papiServiceGetStatusMessage(handle));
3565     ...
3566 }
3567
3568 if (job != NULL)
3569 {
3570     /* process the job object */
3571     attrs = papiJobGetAttributeList(job);
3572     ...
3573     papiJobFree(job);
3574 }
3575
3576 papiServiceDestroy(handle);

```

3576

3577 **See Also**
3578 papiPrinterListJobs, papiJobQuery

3579 **7.10. papiJobGetPrinterName**

3580 **Description**

3581 Get the printer name associated with a job object.

3582 **Syntax**

3583

```
3584           char* papiJobGetPrinterName(  
3585                                    papi_job_t     job );  
3586
```

3587

3588 **Inputs**

3589

3590 job
3591 Handle of the job object.

3592

3593 **Outputs**

3594 none

3595 **Returns**

3596 Pointer to the printer name associated with the job object.

3597 **Example**

3598

```
3599           #include "papi.h"  
3600  
3601           char* printer_name = NULL;  
3602           papi_job_t job = NULL;  
3603           ...  
3604           if (job != NULL)  
3605           {  
3606               /* process the job object */  
3607               printer_name = papiJobGetPrinterName(job);  
3608               ...  
3609               papiJobFree(job);  
3610           }  
3611
```

3612

3613 **See Also**
3614 papiPrinterListJobs, papiJobQuery

3615 **7.11. papiJobGetId**

3616 **Description**

3617 Get the job ID associated with a job object.

3618 **Syntax**

3619

```

3620     int32_t papiJobGetId(
3621         papi_job_t    job );
3622

```

3623

3624 **Inputs**

3625

3626 job

Handle of the job object.

3628

3629 **Outputs**

3630 none

3631 **Returns**

3632 The job ID associated with the job object.

3633 **Example**

3634

```

3635     #include "papi.h"
3636
3637     int32_t job_id;
3638     papi_job_t job = NULL;
3639     ...
3640     if (job != NULL)
3641     {
3642         /* process the job object */
3643         job_id = papiJobGetId(job);
3644         ...
3645         papiJobFree(job);
3646     }
3647

```

3648

3649 **See Also**

3650 papiPrinterListJobs, papiJobQuery

3651 **7.12. papiJobGetJobTicket**3652 **Description**

3653 Get the job ticket associated with a job object.

3654 **Syntax**

3655

```

3656     papi_job_ticket_t* papiJobGetJobTicket(
3657         papi_job_t    job );
3658

```

3659

3660 **Inputs**

3661

3662 job

Handle of the job object.

3663

3664

3665

Outputs

3666

none

3667

Returns

3668

Pointer to the job ticket associated with the job object.

3669

Example

3670

3671

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

3672

```
papi_job_ticket_t* job_ticket = NULL;
```

3673

```
papi_job_t job = NULL;
```

3674

```
...
```

3675

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

3676

```
{
```

3677

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

3678

```
    job_ticket = papiJobGetJobTicket(job);
```

3679

```
    ...
```

3680

```
    papiJobFree(job);
```

3681

```
}
```

3682

3683

3684

See Also

3685

papiPrinterListJobs, papiJobQuery

3686

7.13. papiJobFree

3687

Description

3688

Free a job object.

3689

Syntax

3690

3691

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

3692

3693

3694

3695

Inputs

3696

job

3697

Handle of the job object to free.

3698

3699

3700

Outputs

3701

none

3702

Returns

3703

none

3704

Example

3705

3706

```

3707 #include "papi.h"
3708
3709 papi_status_t status;
3710 papi_service_t handle = NULL;
3711 const char* printer_name = "my-printer";
3712 papi_job_t job = NULL;
3713 ...
3714 status = papiServiceCreate(&handle,
3715                             NULL,
3716                             NULL,
3717                             NULL,
3718                             NULL,
3719                             PAPI_ENCRYPT_NEVER,
3720                             NULL);
3721
3722 if (status != PAPI_OK)
3723 {
3724     /* handle the error */
3725     ...
3726 }
3727
3728 status = papiJobQuery(handle,
3729                       printer_name,
3730                       12,
3731                       &job);
3732
3733 if (status != PAPI_OK)
3734 {
3735     /* handle the error */
3736     fprintf(stderr, "papiJobQuery failed: %s\n",
3737             papiServiceGetStatusMessage(handle));
3738     ...
3739 }
3740
3741 if (job != NULL)
3742 {
3743     /* process the job object */
3744     ...
3745     papiJobFree(job);
3746 }
3747 papiServiceDestroy(handle);

```

3748

3749

See Also

3750

papiJobQuery

3751

7.14. papiJobListFree

3752

Description

3753

Free a list of job objects.

3754

Syntax

3755

```

3756 void papiJobListFree(
3757     papi_job_t* jobs );
3758

```

3759

3760

Inputs

3761

3762 jobs

Pointer to the job object list to free.

3763

3764

3765

Outputs

3766

none

3767

Returns

3768

none

3769

Example

3770

```

3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812

```

```

#include "papi.h"

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

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

status = papiPrinterListJobs(handle,
                             printer_name,
                             NULL,
                             0, 0, 0,
                             &jobs);

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

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

papiServiceDestroy(handle);

```

3813

3814

See Also

3815

papiPrinterListJobs

3816 Chapter 8. Miscellaneous API

3817 8.1. papiStatusString

3818 Description

3819 Get a status string for the specified papi_status_t. The status message returned
3820 from this function may be less detailed than the status message returned from
3821 papiServiceGetStatusMessage (if the print service supports returning more detailed
3822 error messages).

3823 The returned message will be localized in the language of the submitter of the
3824 requestor.

3825 Syntax

3826

```
3827 char* papiStatusString(  
3828     const papi_status_t status );  
3829
```

3830

3831 Inputs

3832

3833 status

3834 The status value to convert to a status string.

3835

3836 Outputs

3837 none

3838 Returns

3839 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3840 value is returned.

3841 Example

3842

```
3843 #include "papi.h"  
3844 papi_status_t status;  
3845 ...  
3846 fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));  
3847  
3848
```

3849

3850 See Also

3851 papiServiceGetStatusMessage

3852 Chapter 9. Attributes

3853 For a summary of the IPP attributes which can be used with the PAPI interface, see:
3854 <ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf>

3855 9.1. Extension Attributes

3856 The following attributes are not currently defined by IPP, but may be used with
3857 this API.

3858 9.1.1. job-ticket-formats-supported

3859 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats
3860 that are supported by the printer. If this attribute is not present, it is assumed that
3861 the printer does not support any job ticket formats.

3862 * *ISSUE: I took the following required attr lists directly from IPP RFC 2911 to use as a starting point. We probably*
3863 *want to add/delete attrs from the lists.*
3864

3865 9.2. Required Job Attributes

3866 The following job attributes *must* be supported to comply with this API standard.
3867 These attributes may be supported by the underlying print server directly, or they
3868 may be mapped by the PAPI library.

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

3869

3870 9.3. Required Printer Attributes

3871 The following printer attributes *must* be supported to comply with this API
3872 standard. These attributes may be supported by the underlying print server
3873 directly, or they may be mapped by the PAPI library.

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

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

3874

3875 **Appendix A. Change History**

3876 **Version 0.5 (August 30, 2002)**

3877

- 3878 • Added job_attrs argument to papiPrinterQuery to support more accurate query
3879 of printer capabilities.
- 3880 • Added management functions papiAttributeDelete, papiJobModify, and
3881 papiPrinterModify.
- 3882 • Added functions papiAttributeListGetValue, papiAttributeListGetString,
3883 papiAttributeListGetInteger, etc.
- 3884 • Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent
3885 with the naming convention (first word after "papi" is the object being operated
3886 upon).
- 3887 • Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.
- 3888 • Made description of authentication more implementation-independent.
- 3889 • Added reference to IPP attributes summary document.
- 3890 • Added result argument to papiPrinterPurgeJobs.
- 3891 • Added "collection attribute" support (PAPI_COLLECTION type).
- 3892 • Changed boolean values to consistently use char. Added PAPI_FALSE and
3893 PAPI_TRUE enum values.

3894

3895 **Version 0.4 (July 19, 2002)**

3896

- 3897 • Made papi_job_t and papi_printer_t opaque handles and added "get" functions
3898 to access the associated information (papiPrinterGetAttributeList,
3899 papiJobGetAttributeList, papiJobGetId, papiJobGetPrinterName,
3900 papiJobGetJobTicket).
- 3901 • Removed variable length argument lists from attribute add functions.
- 3902 • Changed order and name of flag value passed to attribute add functions.
- 3903 • Eliminated indirection in date/time value passed to papiAttributeAddDatetime.
- 3904 • Added message argument to papiPrinterPause.

3905

3906 **Version 0.3 (June 24, 2002)**

3907

- 3908 • Converted to DocBook format from Microsoft Word
- 3909 • Major rewrite, including:
 - 3910 • Changed how printer names are described in "Model/Printer"
 - 3911 • Changed fixed length strings to pointers in numerous structures/sections
 - 3912 • Redefined attribute/value structures and associated API descriptions
 - 3913 • Changed list/query functions to return "objects"

- 3914 • Rewrote "Attributes API" chapter
- 3915 • Changed many function definitions to pass NULL-terminated arrays of
- 3916 pointers instead of a separate count argument
- 3917 • Changed papiJobSubmit to take an attribute list structure as input instead of a
- 3918 formatted string

3919

3920

3921 **Version 0.2 (April 17, 2002)**

3922

- 3923 • Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)
- 3924 • Filled in "Encryption" section and added information about encryption in "Object
- 3925 Identification" section
- 3926 • Added "short_name" field in "Object Identification" section
- 3927 • Added "Job Ticket (papi_job_ticket_t)" section
- 3928 • Added papiPrinterPause
- 3929 • Added papiPrinterResume
- 3930 • Added papiPurgeJobs
- 3931 • Added optional job_ticket argument to papiJobSubmit
- 3932 • Added optional passing of filenames by URI to papiJobSubmit
- 3933 • Added papiHoldJob
- 3934 • Added papiReleaseJob
- 3935 • Added papiRestartJob

3936

3937 **Version 0.1 (April 3, 2002)**

3938

- 3939 • Original draft version

3940

3941

3942

3943

3944

3945

<i>End of Document</i>
