

1

2

Open Standard Print API (PAPI)

3

Version 0.4 (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 0.4 (DRAFT)**

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

14 Version 0.3 (DRAFT) Edition

15 Copyright © 2002 by Free Standards Group

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

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).....	6
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. papiPrinterPause.....	25
58	5.5. papiPrinterResume.....	27
59	5.6. papiPrinterPurgeJobs	28
60	5.7. papiPrinterListJobs	29
61	5.8. papiPrinterGetAttributeList	31
62	5.9. papiPrinterFree	32
63	5.10. papiPrinterListFree.....	33
64	6. Attributes API.....	35
65	6.1. papiAttributeAdd	35
66	6.2. papiAttributeAddString	36

67	6.3. papiAttributeAddInteger	37
68	6.4. papiAttributeAddBoolean.....	38
69	6.5. papiAttributeAddRange.....	40
70	6.6. papiAttributeAddResolution.....	41
71	6.7. papiAttributeAddDatetime.....	42
72	6.8. papiAttributeListFree.....	43
73	6.9. papiAttributeListFind	44
74	6.10. papiAttributeListGetNext.....	45
75	7. Job API	47
76	7.1. papiJobSubmit.....	47
77	7.2. papiJobValidate.....	48
78	7.3. papiJobQuery	50
79	7.4. papiJobCancel	52
80	7.5. papiJobHold	53
81	7.6. papiJobRelease	54
82	7.7. papiJobRestart	56
83	7.8. papiJobGetAttributeList	57
84	7.9. papiJobGetPrinterName	58
85	7.10. papiJobGetId	59
86	7.11. papiJobGetJobTicket.....	60
87	7.12. papiJobFree.....	60
88	7.13. papiJobListFree	62
89	8. Miscellaneous API	64
90	8.1. papiStatusString.....	64
91	9. Attributes	65
92	9.1. Extension Attributes.....	65
93	9.1.1. job-ticket-formats-supported.....	65
94	9.2. Required Job Attributes	65
95	9.3. Required Printer Attributes.....	65
96	A. Change History	67

97 **Chapter 1. Introduction**

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

105 This version of the document describes stage 1 and stage 2 of the Open Standard
106 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.)

107

108

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

111 Chapter 2. Print System Model

112 2.1. Introduction

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

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

- 119 • Object Definition: What objects are part of the model?
- 120 • Object Naming: How is each object identified/named?
- 121 • Object Relationships: What are the associations and relationships between the
122 objects?

123

124 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	

125

126

127 2.2. Model

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

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

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

138 2.2.1. Print Service

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

143 2.2.2. Printer

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

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

152 **2.2.3. Job**

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

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

159 **2.3. Security**

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

162 **2.3.1. Authentication**

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

169 If the user name and password are not passed on the API call, the call may fail with
170 an error code indicating a security problem (e.g. PAPI_NOT_AUTHENTICATED).

171 See RFC 2616 and RFC 2617 for further details about HTTP security.

172 **2.3.2. Authorization**

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

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

181 **2.3.3. Encryption**

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

187 Chapter 3. Common Structures

188 3.1. Conventions

189

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

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

195

196 3.2. Service Object (papi_service_t)

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

200
201

```
typedef void* papi_service_t;
```

202

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

204

```
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;
```

205
206
207
208
209
210
211

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

214 3.3. Attributes and Values

215 These are the structures defining how attributes and values are passed to and from
216 PAPI.

217

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

218
219
220
221
222
223
224
225
226
227

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

229

```
/* Attribute Value */  
typedef union  
{  
    char* string;      /* PAPI_STRING value */  
  
    int integer;      /* PAPI_INTEGER value */  
  
    char boolean;     /* PAPI_BOOLEAN value */  
  
    struct             /* PAPI_RANGE value */  
    {  
        int lower;
```

230
231
232
233
234
235
236
237
238
239
240


```

241         int upper;
242     } range;
243
244     struct          /* PAPI_RESOLUTION value */
245     {
246         int xres;
247         int yres;
248     } resolution;
249
250     time_t datetime; /* PAPI_DATETIME value */
251 } papi_attribute_value_t;
252
253
254 /* Attribute and Values */
255 typedef struct
256 {
257     char* name;          /* attribute name */
258     papi_attribute_value_type_t type; /* type of values */
259     papi_attribute_value_t** values; /* list of values */
260 } papi_attribute_t;
261
262 /* Attribute add flags */
263 #define PAPI_ATTR_APPEND 0x0001 /* Add values to attr */
264 #define PAPI_ATTR_REPLACE 0x0002 /* Delete existing
265                                values then add new ones */
266 #define PAPI_ATTR_EXCL 0x0004 /* Fail if attr exists */

```

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

268 3.4. Job Object (papi_job_t)

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

273 3.5. Printer Object (papi_printer_t)

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

278 3.6. Job Ticket (papi_job_ticket_t)

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

```

283 /* Job Ticket Format */
284 typedef enum
285 {
286     PAPI_JT_FORMAT_JDF = 0,          /* Job Definition Format */
287 } papi_jt_format_t;
288

```

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

```

290 /* Job Ticket */
291 typedef struct papi_job_ticket_s
292 {
293     papi_jt_format_t format; /* Format of job ticket */
294     char* ticket_data; /* Buffer containing the job
295                       ticket data. If NULL,
296                       uri must be specified */
297     char* uri; /* URI of the file containing
298               the job ticket data. If
299               ticket_data is specified, then
300               uri is ignored. */
301 } papi_job_ticket_t;
302

```

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

304 3.7. Status (`papi_status_t`)

```

305     typedef enum
306     {
307         PAPI_OK = 0x0000,
308         PAPI_OK_SUBST,
309         PAPI_OK_CONFLICT,
310         PAPI_OK_IGNORED_SUBSCRIPTIONS,
311         PAPI_OK_IGNORED_NOTIFICATIONS,
312         PAPI_OK_TOO_MANY_EVENTS,
313         PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
314         PAPI_REDIRECTION_OTHER_SITE = 0x300,
315         PAPI_BAD_REQUEST = 0x0400,
316         PAPI_FORBIDDEN,
317         PAPI_NOT_AUTHENTICATED,
318         PAPI_NOT_AUTHORIZED,
319         PAPI_NOT_POSSIBLE,
320         PAPI_TIMEOUT,
321         PAPI_NOT_FOUND,
322         PAPI_GONE,
323         PAPI_REQUEST_ENTITY,
324         PAPI_REQUEST_VALUE,
325         PAPI_DOCUMENT_FORMAT,
326         PAPI_ATTRIBUTES,
327         PAPI_URI_SCHEME,
328         PAPI_CHARSET,
329         PAPI_CONFLICT,
330         PAPI_COMPRESSION_NOT_SUPPORTED,
331         PAPI_COMPRESSION_ERROR,
332         PAPI_DOCUMENT_FORMAT_ERROR,
333         PAPI_DOCUMENT_ACCESS_ERROR,
334         PAPI_ATTRIBUTES_NOT_SETTABLE,
335         PAPI_IGNORED_ALL_SUBSCRIPTIONS,
336         PAPI_TOO_MANY_SUBSCRIPTIONS,
337         PAPI_IGNORED_ALL_NOTIFICATIONS,
338         PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
339         PAPI_INTERNAL_ERROR = 0x0500,
340         PAPI_OPERATION_NOT_SUPPORTED,
341         PAPI_SERVICE_UNAVAILABLE,
342         PAPI_VERSION_NOT_SUPPORTED,
343         PAPI_DEVICE_ERROR,
344         PAPI_TEMPORARY_ERROR,
345         PAPI_NOT_ACCEPTING,
346         PAPI_PRINTER_BUSY,
347         PAPI_ERROR_JOB_CANCELLED,
348         PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
349         PAPI_PRINTER_IS_DEACTIVATED,
350         PAPI_BAD_ARGUMENT
351     } papi_status_t;

```

353 NOTE: If a Particular implementation of PAPI does not support a requested
354 function, `PAPI_OPERATION_NOT_SUPPORTED` must be returned from that
355 function.

356 3.8. List Filter (`papi_filter_t`)

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

```

360     typedef enum
361     {
362         PAPI_FILTER_BITMASK = 0
363         /* future filter types may be added here */
364     } papi_filter_type_t;
365
366     typedef struct
367     {
368         papi_filter_type_t  type; /* Type of filter specified */
369
370         union
371         {
372             unsigned int  mask; /* PAPI_FILTER_BITMASK */
373
374             /* future filter types may be added here */
375         } u;
376     } papi_filter_t;
377

```

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

```

380 enum
381 {
382     PAPI_PRINTER_LOCAL = 0x0000,      /* Local printer or class */
383     PAPI_PRINTER_CLASS = 0x0001,     /* Printer class */
384     PAPI_PRINTER_REMOTE = 0x0002,    /* Remote printer or class */
385     PAPI_PRINTER_BW = 0x0004,        /* Can do B&W printing */
386     PAPI_PRINTER_COLOR = 0x0008,     /* Can do color printing */
387     PAPI_PRINTER_DUPLEX = 0x0010,    /* Can do duplexing */
388     PAPI_PRINTER_STAPLE = 0x0020,    /* Can staple output */
389     PAPI_PRINTER_COPIES = 0x0040,    /* Can do copies */
390     PAPI_PRINTER_COLLATE = 0x0080,   /* Can collage copies */
391     PAPI_PRINTER_PUNCH = 0x0100,     /* Can punch output */
392     PAPI_PRINTER_COVER = 0x0200,     /* Can cover output */
393     PAPI_PRINTER_BIND = 0x0400,      /* Can bind output */
394     PAPI_PRINTER_SORT = 0x0800,      /* Can sort output */
395     PAPI_PRINTER_SMALL = 0x1000,     /* Can do Letter/Legal/A4 */
396     PAPI_PRINTER_MEDIUM = 0x2000,    /* Can do Tabloid/B/C/A3/A2 */
397     PAPI_PRINTER_LARGE = 0x4000,     /* Can do D/E/A1/A0 */
398     PAPI_PRINTER_VARIABLE = 0x8000,  /* Can do variable sizes */
399     PAPI_PRINTER_IMPLICIT = 0x10000, /* Implicit class */
400     PAPI_PRINTER_DEFAULT = 0x20000,  /* Default printer on network */
401     PAPI_PRINTER_OPTIONS = 0xffffc,   /* ~(CLASS | REMOTE | IMPLICIT) */
402 };
403

```

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

405 Chapter 4. Service API

406 4.1. papiServiceCreate

407 Description

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

412 Syntax

413

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

423

424 Inputs

425

426 service_name

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

431 user_name

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

435 password

436 (optional) Points to the password to be used to authenticate the user to the
437 print service.

438 authCB

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

444 encryption

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

446 app_data

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

449

450 **Outputs**

451

452 handle

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

456

457 **Returns**

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

460 **Example**

461

```

462 #include "papi.h"
463
464 papi_status_t status;
465 papi_service_t handle = NULL;
466 const char* service_name = "ipp:/printserv:631";
467 const char* user_name = "pappy";
468 const char* password = "goober";
469 ...
470 status = papiServiceCreate(&handle,
471                            service_name,
472                            user_name,
473                            password,
474                            NULL,
475                            PAPI_ENCRYPT_IF_REQUESTED,
476                            NULL);
477
478 if (status != PAPI_OK)
479 {
480     /* handle the error */
481     fprintf(stderr, "papiServiceCreate failed: %s\n",
482            papiStatusString(status));
483     if (handle != NULL)
484     {
485         fprintf(stderr, "    details: %s\n",
486                papiServiceGetStatusMessage(handle));
487     }
488     ...
489 }
490 ...
491 papiServiceDestroy(handle);

```

492

493 **See Also**

494 papiServiceDestroy, papiServiceGetStatusMessage, papiServiceSetUsername,
495 papiServiceSetPassword, papiServiceSetEncryption, papiServiceSetAuthCB

496 **4.2. papiServiceDestroy**

497 **Description**

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

501 **Syntax**

502

```
503           void papiServiceDestroy(  
504                 papi_service_t handle );  
505
```

506

507 **Inputs**

508

509 handle

510 The print service handle to be destroyed.

511

512 **Outputs**

513 none

514 **Returns**

515 none

516 **Example**

517

```
518           #include "papi.h"  
519  
520           papi_status_t status;  
521           papi_service_t handle = NULL;  
522           const char* service_name = "ipp://printserv:631";  
523           const char* user_name = "pappy";  
524           const char* password = "goober";  
525           ...  
526           status = papiServiceCreate(&handle,  
527                                     service_name,  
528                                     user_name,  
529                                     password,  
530                                     NULL,  
531                                     PAPI_ENCRYPT_IF_REQUESTED,  
532                                     NULL);  
533  
534           if (status != PAPI_OK)  
535           {  
536                 /* handle the error */  
537                 ...  
538           }  
539           ...  
540           papiServiceDestroy(handle);
```

541

542 **See Also**

543 papiServiceCreate

544 **4.3. papiServiceSetUsername**

545 **Description**

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

549 **Syntax**

550

```

551     papi_status_t papiServiceSetUsername (
552         papi_service_t handle,
553         const char* user_name );
554

```

555

556 **Inputs**

557

558 handle

559 Handle to the print service to update.

560 user_name

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

564

565 **Outputs**

566 handle is updated.

567 **Returns**

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

570 **Example**

571

```

572     #include "papi.h"
573
574     papi_status_t status;
575     papi_service_t handle = NULL;
576     const char* user_name = "pappy";
577     ...
578     status = papiServiceCreate(&handle,
579                               NULL,
580                               NULL,
581                               NULL,
582                               NULL,
583                               PAPI_ENCRYPT_IF_REQUESTED,
584                               NULL);
585
586     if (status != PAPI_OK)
587     {
588         /* handle the error */
589         ...
590     }
591
592     status = papiServiceSetUsername(handle, user_name);
593     if (status != PAPI_OK)
594     {
595         /* handle the error */
596         fprintf(stderr, "papiServiceSetUsername failed: %s\n",
597               papiServiceGetStatusMessage(handle));
598         ...
599     }
600     ...
601     papiServiceDestroy(handle);

```

602

603 **See Also**

604 papiServiceCreate, papiServiceSetPassword, papiServiceGetStatusMessage

605 **4.4. papiServiceSetPassword**606 **Description**

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

610 **Syntax**

611

```
612 papi_status_t papiServiceSetPassword(
613     papi_service_t handle,
614     const char* password );
615
```

616

617 **Inputs**

618

619 handle

620 Handle to the print service to update.

621 password

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

623

624 **Outputs**

625 handle is updated.

626 **Returns**

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

629 **Example**

630

```
631 #include "papi.h"
632
633 papi_status_t status;
634 papi_service_t handle = NULL;
635 const char* password = "goober";
636 ...
637 status = papiServiceCreate(&handle,
638     NULL,
639     NULL,
640     NULL,
641     NULL,
642     PAPI_ENCRYPT_IF_REQUESTED,
643     NULL);
644
645 if (status != PAPI_OK)
646 {
647     /* handle the error */
648     ...
649 }
650
651 status = papiServiceSetPassword(handle, password);
652 if (status != PAPI_OK)
653 {
654     /* handle the error */
655     fprintf(stderr, "papiServiceSetPassword failed: %s\n",
656         papiServiceGetStatusMessage(handle));
657     ...
658 }
```



```
659     papiServiceDestroy(handle);
660
```

661

662 **See Also**

663 papiServiceCreate, papiServiceSetUsername, papiServiceGetStatusMessage

664 4.5. papiServiceSetEncryption

665 **Description**

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

667 **Syntax**

668

```
669     papi_status_t papiServiceSetEncryption(
670         papi_service_t handle,
671         const papi_encryption_t encryption );
672
```

673

674 **Inputs**

675

676 handle

677 Handle to the print service to update.

678 encryption

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

680

681 **Outputs**

682 handle is updated.

683 **Returns**

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

686 **Example**

687

```
688     #include "papi.h"
689
690     papi_status_t status;
691     papi_service_t handle = NULL;
692     ...
693     status = papiServiceCreate(&handle,
694         NULL,
695         NULL,
696         NULL,
697         NULL,
698         PAPI_ENCRYPT_IF_REQUESTED,
699         NULL);
700
701     if (status != PAPI_OK)
702     {
703         /* handle the error */
704         ...
705     }
706
707     status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);
708     if (status != PAPI_OK)
```

```

708
709     {
710         /* handle the error */
711         fprintf(stderr, "papiServiceSetEncryption failed: %s\n",
712                papiServiceGetStatusMessage(handle));
713         ...
714     }
715     ...
716     papiServiceDestroy(handle);

```

717

718 **See Also**

719 papiServiceCreate, papiServiceGetStatusMessage

720 **4.6. papiServiceSetAuthCB**721 **Description**

722 Set the authorization callback function in the print service handle to be used in
 723 subsequent calls.

724 **Syntax**

725

```

726 papi_status_t papiServiceSetAuthCB(
727     papi_service_t handle,
728     const int (*authCB)(papi_service_t svc) );
729

```

730

731 **Inputs**

732

733 handle

734 Handle to the print service to update.

735 authCB

736 Points to a callback function to be used in authenticating the user to the print
 737 service if no password was supplied (or user input is required). A NULL value
 738 indicates that no callback should be made. The callback function should return
 739 0 if the request is to be cancelled and non-zero if new authentication
 740 information has been set.

741

742 **Outputs**

743 handle is updated.

744 **Returns**

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

747 **Example**

748

```

749 #include "papi.h"
750
751 extern int get_password(papi_service_t handle);
752 papi_status_t status;
753 papi_service_t handle = NULL;

```

```

754     ...
755     status = papiServiceCreate(&handle,
756                               NULL,
757                               NULL,
758                               NULL,
759                               NULL,
760                               PAPI_ENCRYPT_IF_REQUESTED,
761                               NULL);
762
763     if (status != PAPI_OK)
764     {
765         /* handle the error */
766         ...
767     }
768
769     status = papiServiceSetAuthCB(handle, get_password);
770     if (status != PAPI_OK)
771     {
772         /* handle the error */
773         fprintf(stderr, "papiServiceSetAuthCB failed: %s\n",
774                papiServiceGetStatusMessage(handle));
775         ...
776     }
777     ...
778     papiServiceDestroy(handle);

```

779

780 **See Also**

781 papiServiceCreate, papiServiceGetStatusMessage

782 **4.7. papiServiceSetAppData**783 **Description**

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

787 **Syntax**

788

```

789 papi_status_t papiServiceSetAppData (
790     papi_service_t handle,
791     const void* app_data );
792

```

793

794 **Inputs**

795

796 handle

797 Handle to the print service to update.

798 app_data

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

801

802 **Outputs**

803 handle is updated.

804

Returns

805

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

806

807

Example

808

809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839

```
#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);
```

840

841

See Also

842

papiServiceCreate, papiServiceGetStatusMessage

843 4.8. papiServiceGetServicename

844

Description

845

Get the service name associated with the print service handle.

846

Syntax

847

848

849

850

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

851

852

Inputs

853

854

handle

855

Handle to the print service.

856

857

Outputs

858

none

859

Returns

860

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

861

Example

862

```

863 #include "papi.h"
864
865 papi_status_t status;
866 papi_service_t handle = NULL;
867 char* service_name = NULL;
868 ...
869 service_name = papiServiceGetServicename(handle);
870 if (service_name != NULL)
871 {
872     /* use the returned name */
873     ...
874 }
875 ...
876 papiServiceDestroy(handle);
877

```

878

879

See Also

880

papiServiceCreate

881 **4.9. papiServiceGetUsername**

882

Description

883

Get the user name associated with the print service handle.

884

Syntax

885

```

886 char* papiServiceGetUsername(
887     papi_service_t handle );
888

```

889

890

Inputs

891

892 handle

893

Handle to the print service.

894

895

Outputs

896

none

897

Returns

898

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

899

Example

900

```

901     #include "papi.h"
902
903     papi_status_t status;
904     papi_service_t handle = NULL;
905     char* user_name = NULL;
906     ...
907     user_name = papiServiceGetUsername(handle);
908     if (user_name != NULL)
909     {
910         /* use the returned name */
911         ...
912     }
913     ...
914     papiServiceDestroy(handle);
915

```

916

See Also

papiServiceCreate, papiServiceSetUsername

4.10. papiServiceGetPassword

Description

Get the user password associated with the print service handle.

Syntax

923

```

924     char* papiServiceGetPassword(
925         papi_service_t handle );
926

```

927

Inputs

929

930 handle

Handle to the print service.

932

Outputs

none

935

Returns

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

937

Example

938

```

939     #include "papi.h"
940
941     papi_status_t status;
942     papi_service_t handle = NULL;
943     char* password = NULL;
944     ...
945     password = papiServiceGetPassword(handle);
946     if (password != NULL)
947     {
948         /* use the returned password */
949         ...
950     }
951     ...
952     papiServiceDestroy(handle);
953

```

954

955

See Also

956

papiServiceCreate, papiServiceSetPassword

957

4.11. papiServiceGetEncryption

958

Description

959

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

960

Syntax

961

962

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

963

964

965

966

Inputs

967

968

handle

969

Handle to the print service.

970

971

Outputs

972

none

973

Returns

974

The type of encryption associated with the print service handle.

975

Example

976

977

```
#include "papi.h"

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

978

979

980

981

982

983

984

985

986

987

988

989

See Also

990

papiServiceCreate, papiServiceSetEncryption

991

4.12. papiServiceGetAppData

992

Description

993

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

994

handle.

995 **Syntax**

996

```
997           void* papiServiceGetAppData(
998                     papi_service_t handle );
999
```

1000

1001 **Inputs**

1002

1003 handle

 Handle to the print service.

1005

1006 **Outputs**

1007 none

1008 **Returns**

1009 A pointer to the application-specific data associated with the print service handle.

1010 **Example**

1011

```
1012           #include "papi.h"
1013
1014           papi_status_t status;
1015           papi_service_t handle = NULL;
1016           char* app_data = NULL;
1017           ...
1018           app_data = (char*)papiServiceGetAppData(handle);
1019           if (app_data != NULL)
1020           {
1021               /* use the returned application data */
1022               ...
1023           }
1024           ...
1025           papiServiceDestroy(handle);
1026
```

1027

1028 **See Also**

1029 papiServiceCreate, papiServiceSetAppData

1030 **4.13. papiServiceGetStatusMessage**

1031 **Description**

1032 Get the message associated with the status of the last operation performed. The
 1033 status message returned from this function may be more detailed than the status
 1034 message returned from papiStatusString (if the print service supports returning
 1035 more detailed error messages).

1036 The returned message will be localized in the language of the submitter of the
 1037 original operation.

1038 **Syntax**

1039

```
1040           const char* papiServiceGetStatusMessage(
```



```
1041         const papi_service_t handle );
1042
```

1043

1044 **Inputs**

1045

1046 handle

1047 Handle to the print service.

1048

1049 **Outputs**

1050 none

1051 **Returns**

1052 Pointer to the message associated with the status of the last operation performed.

1053 **Example**

1054

```
1055 #include "papi.h"
1056
1057 papi_status_t status;
1058 papi_service_t handle = NULL;
1059 const char* user_name = "pappy";
1060 ...
1061 status = papiServiceCreate(&handle,
1062                          NULL,
1063                          NULL,
1064                          NULL,
1065                          NULL,
1066                          PAPI_ENCRYPT_IF_REQUESTED,
1067                          NULL);
1068
1069 if (status != PAPI_OK)
1070 {
1071     /* handle the error */
1072     ...
1073 }
1074
1075 status = papiServiceSetUsername(handle, user_name);
1076 if (status != PAPI_OK)
1077 {
1078     /* handle the error */
1079     fprintf(stderr, "papiServiceSetUsername failed: %s\n",
1080            papiServiceGetStatusMessage(handle));
1081     ...
1082 }
1083 ...
1084 papiServiceDestroy(handle);
```

1085

1086 **See Also**

1087 papiStatusString

1088 Chapter 5. Printer API

1089 5.1. Usage

1090 The papiPrinterQuery function queries all/some of the attributes of a printer
1091 object. It returns a list of printer attributes. A successful call to papiPrinterQuery is
1092 typically followed by code which examines and processes the returned attributes.
1093 The using program would then call papiPrinterFree to delete the returned results.

1094 Printers can be found via calls to papiPrintersList. A successful call to
1095 papiPrintersList is typically followed by code to iterate through the list of returned
1096 printers, possibly querying each (papiPrinterQuery) for further information (e.g. to
1097 restrict what printers get displayed for a particular user/request). The using
1098 program would then call papiPrinterListFree to free the returned results.

1099 5.2. papiPrintersList

1100 Description

1101 List all printers known by the print service which match the specified filter.

1102 Depending on the functionality of the target service's "printer directory", the
1103 returned list may be limited to only printers managed by a particular server or it
1104 may include printers managed by other servers.

1105 Syntax

1106

```
1107 papi_status_t papiPrintersList(  
1108             papi_service_t   handle,  
1109             const char*      requested_attrs[],  
1110             const papi_filter_t* filter,  
1111             papi_printer_t** printers );  
1112
```

1113

1114 Inputs

1115

1116 handle

1117 Handle to the print service to use.

1118 requested_attrs

1119 (optional) NULL terminated array of attribute names to be queried. If NULL is
1120 passed then all available attributes should be returned.

1121 filter

1122 (optional) Pointer to a filter to limit the number of printers returned on the list
1123 request. See Section 3.8 for details. If NULL is passed then all known printers
1124 are listed.

1125

1126 Outputs

1127

1128 printers

1129 List of printer objects that matched the filter criteria.

1130

1131 **Returns**

1132 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1133 value is returned.

1134 **Example**

1135

```

1136 #include "papi.h"
1137
1138 int i;
1139 papi_status_t status;
1140 papi_service_t handle = NULL;
1141 const char* service_name = "ipp://printserv:631";
1142 const char* user_name = "pappy";
1143 const char* password = "goober";
1144 const char* req_attrs[] =
1145 {
1146     "printer-name",
1147     "printer-location",
1148     NULL
1149 };
1150 const papi_filter_t filter =
1151     PAPI_PRINTER_BW | PAPI_PRINTER_DUPLEX;
1152 papi_printer_t* printers = NULL;
1153 ...
1154 status = papiServiceCreate(&handle,
1155                             service_name,
1156                             user_name,
1157                             password,
1158                             NULL,
1159                             PAPI_ENCRYPT_IF_REQUESTED,
1160                             NULL);
1161
1162 if (status != PAPI_OK)
1163 {
1164     /* handle the error */
1165     ...
1166 }
1167
1168 status = papiPrinterList(handle,
1169                          req_attrs,
1170                          filter,
1171                          &printers);
1172
1173 if (status != PAPI_OK)
1174 {
1175     /* handle the error */
1176     fprintf(stderr, "papiPrinterList failed: %s\n",
1177            papiServiceGetStatusMessage(handle));
1178     ...
1179 }
1180
1181 if (printers != NULL)
1182 {
1183     for (i=0; printers[i] != NULL; i++)
1184     {
1185         /* process the printer object */
1186         ...
1187     }
1188     papiPrinterListFree(printers);
1189 }
1190
1191 papiServiceDestroy(handle);

```

1191

1192 **See Also**

1193 papiPrinterListFree, papiPrinterQuery

1194 **5.3. papiPrinterQuery**1195 **Description**

1196 Queries some or all the attributes of the specified printer object. This includes
 1197 attributes representing the capabilities of the printer, which the caller may use to
 1198 determine which print options to present to the user. How the attributes are
 1199 obtained (e.g. from a static database, from a dialog with the hardware, from a dialog
 1200 with a driver, etc.) is up to the implementer of the API and is beyond the scope of
 1201 this standard.

1202 **Syntax**

1203

```

1204 papi_status_t papiPrinterQuery(
1205             papi_service_t   handle,
1206             const char*      name,
1207             const char*      requested_attrs[],
1208             papi_printer_t*  printer );
1209

```

1210

1211 **Inputs**

1212

1213 handle

1214 Handle to the print service to use.

1215 name

1216 The name or URI of the printer to query.

1217 requested_attrs

1218 (optional) NULL terminated array of attributes to be queried. If NULL is
 1219 passed then all attributes are queried. (NOTE: The printer may return more
 1220 attributes than you requested. This is merely an advisory request that may
 1221 reduce the amount of data returned if the printer/server supports it.)

1222

1223 **Outputs**

1224

1225 printer

1226 Pointer to a printer object containing the requested attributes.

1227

1228 **Returns**

1229 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1230 value is returned.

1231 **Example**

1232

```

1233 #include "papi.h"
1234

```

```

1235 papi_status_t status;
1236 papi_service_t handle = NULL;
1237 const char* service_name = "ipp://printserv:631";
1238 const char* user_name = "pappy";
1239 const char* password = "goober";
1240 const char* printer_name = "my-printer";
1241 const char* req_attrs[] =
1242 {
1243     "printer-name",
1244     "printer-location",
1245     "printer-state",
1246     "printer-state-reasons",
1247     "printer-state-message",
1248     NULL
1249 };
1250 papi_printer_t printer = NULL;
1251 ...
1252 status = papiServiceCreate(&handle,
1253                             service_name,
1254                             user_name,
1255                             password,
1256                             NULL,
1257                             PAPI_ENCRYPT_IF_REQUESTED,
1258                             NULL);
1259
1260 if (status != PAPI_OK)
1261 {
1262     /* handle the error */
1263     ...
1264 }
1265
1266 status = papiPrinterQuery(handle,
1267                             printer_name,
1268                             req_attrs,
1269                             &printer);
1270
1271 if (status != PAPI_OK)
1272 {
1273     /* handle the error */
1274     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1275             papiServiceGetStatusMessage(handle));
1276     ...
1277 }
1278
1279 if (printer != NULL)
1280 {
1281     /* process the printer object */
1282     ...
1283     papiPrinterFree(printer);
1284 }
1285 papiServiceDestroy(handle);

```

1286

1287

See Also

1288

papiPrinterList, papiPrinterFree

1289

5.4. papiPrinterPause

1290

Description

1291

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.

1292

1293

1294

1295

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

1296

1297

Syntax

1298

1299

```

1300 papi_status_t papiPrinterPause(
1301             papi_service_t handle,
1302             const char* name,
1303             const char* message );

```

1300

1301

1302

1303

1304

1305 **Inputs**

1306

1307 handle

1308 Handle to the print service to use.

1309 name

1310 The name or URI of the printer to operate on.

1311 message

1312 (optional) An explanatory message to be associated with the paused printer.
 1313 This message may be ignored if the underlying print system does not support
 1314 associating a message with a paused printer.

1315

1316 **Outputs**

1317 none

1318 **Returns**

1319 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1320 value is returned.

1321 **Example**

1322

```

1323 #include "papi.h"
1324
1325 papi_status_t status;
1326 papi_service_t handle = NULL;
1327 const char* service_name = "ipp://printserv:631";
1328 const char* user_name = "pappy";
1329 const char* password = "goober";
1330 const char* printer_name = "my-printer";
1331 ...
1332 status = papiServiceCreate(&handle,
1333                             service_name,
1334                             user_name,
1335                             password,
1336                             NULL,
1337                             PAPI_ENCRYPT_IF_REQUESTED,
1338                             NULL);
1339
1340 if (status != PAPI_OK)
1341 {
1342     /* handle the error */
1343     ...
1344 }
1345
1346 status = papiPrinterPause(handle, printer_name, NULL);
1347 if (status != PAPI_OK)
1348 {
1349     /* handle the error */
1350     fprintf(stderr, "papiPrinterPause failed: %s\n",
1351             papiServiceGetStatusMessage(handle));
1352     ...
1353 }
1354 ...
1355 papiServiceDestroy(handle);

```

1356

1357 **See Also**

1358 papiPrinterResume

1359 **5.5. papiPrinterResume**1360 **Description**

1361 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the
 1362 effects of papiPrinterPause). This operation is optional and may not be supported
 1363 by all printers/servers, but it must be supported if papiPrinterPause is supported.

1364 **Syntax**

1365

```
1366 papi_status_t papiPrinterResume(  
1367             papi_service_t   handle,  
1368             const char*      name );  
1369
```

1370

1371 **Inputs**

1372

1373 handle

Handle to the print service to use.

1374

1375 name

The name or URI of the printer to operate on.

1376

1377

1378 **Outputs**

1379 none

1380 **Returns**

1381 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1382 value is returned.

1383 **Example**

1384

```
1385 #include "papi.h"  
1386  
1387 papi_status_t status;  
1388 papi_service_t handle = NULL;  
1389 const char* service_name = "ipp://printserv:631";  
1390 const char* user_name = "pappy";  
1391 const char* password = "goober";  
1392 const char* printer_name = "my-printer";  
1393 ...  
1394 status = papiServiceCreate(&handle,  
1395                          service_name,  
1396                          user_name,  
1397                          password,  
1398                          NULL,  
1399                          PAPI_ENCRYPT_IF_REQUESTED,  
1400                          NULL);  
1401  
1402 if (status != PAPI_OK)  
1403 {  
1404     /* handle the error */  
1405     ...  
1406 }  
1407  
1408 status = papiPrinterPause(handle, printer_name);  
1409 if (status != PAPI_OK)  
1410 {  
1411     /* handle the error */  
1412     fprintf(stderr, "papiPrinterPause failed: %s\n",  
            papiServiceGetStatusMessage(handle));
```

```

1413     ...
1414 }
1415     ...
1416 status = papiPrinterResume(handle, printer_name);
1417 if (status != PAPI_OK)
1418 {
1419     /* handle the error */
1420     fprintf(stderr, "papiPrinterResume failed: %s\n",
1421            papiServiceGetStatusMessage(handle));
1422     ...
1423 }
1424
1425 papiServiceDestroy(handle);
1426

```

1427

See Also

1429 papiPrinterPause

5.6. papiPrinterPurgeJobs**1431 Description**

1432 Remove all jobs from the specified printer object regardless of their states. This
 1433 includes removing jobs that have completed and are being kept for history (if any).
 1434 This operation is optional and may not be supported by all printers/servers.

1435 Syntax

1436

```

1437 papi_status_t papiPrinterPurgeJobs (
1438     papi_service_t handle,
1439     const char* name );
1440

```

1441

1442 Inputs

1443

1444 handle

Handle to the print service to use.

1446 name

The name or URI of the printer to operate on.

1448

1449 Outputs

1450 none

1451 Returns

1452 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1453 value is returned.

1454 Example

1455

```

1456 #include "papi.h"
1457
1458 papi_status_t status;
1459 papi_service_t handle = NULL;
1460 const char* service_name = "ipp://printserv:631";

```



```

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

```

1489

1490 **See Also**1491 `papiJobCancel`1492 **5.7. papiPrinterListJobs**1493 **Description**

1494 List print job(s) associated with the specified printer.

1495 **Syntax**

1496

```

1497     papi_status_t papiPrinterListJobs(
1498         papi_service_t    handle,
1499         const char*       printer,
1500         const char*       requested_attrs[],
1501         const int         type_mask,
1502         const int         max_num_jobs,
1503         papi_job_t**      jobs );
1504

```

1505

1506 **Inputs**

1507

1508 `handle`

1509 Handle to the print service to use.

1510 `requested_attrs`

1511 (optional) NULL terminated array of attributes to be queried. If NULL is
 1512 passed then all available attributes are queried. (NOTE: The printer may return
 1513 more attributes than you requested. This is merely an advisory request that
 1514 may reduce the amount of data returned if the printer/server supports it.)

1515 type_mask

1516 A bit mask which determines what jobs will get returned. The following
1517 constants can be bitwise-OR-ed together to select which types of jobs to list:

```
1518 #define PAPI_LIST_JOBS_OTHERS      0x0001 /* return jobs other than
1519                                     those submitted by the
1520                                     user name assoc with
1521                                     the handle */
1522 #define PAPI_LIST_JOBS_COMPLETED  0x0002 /* return completed jobs */
1523 #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
1524                                     jobs */
1525 #define PAPI_LIST_JOBS_ALL        0xFFFF /* return all jobs */
1526
```

1527

1528 max_num_jobs

1529 Limit to the number of jobs returned. If 0 is passed, then there is no limit on
1530 the number of jobs which may be returned.

1531

1532 **Outputs**

1533

1534 jobs

1535 List of job objects returned.

1536

1537 **Returns**

1538 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1539 value is returned.

1540 **Example**

1541

```
1542 #include "papi.h"
1543
1544 int i;
1545 papi_status_t status;
1546 papi_service_t handle = NULL;
1547 const char* printer_name = "my-printer";
1548 papi_job_t* jobs = NULL;
1549 const char* job_attrs[] =
1550 {
1551     "job-id",
1552     "job-name",
1553     "job-originating-user-name",
1554     "job-state",
1555     "job-state-reasons",
1556     NULL
1557 };
1558 ...
1559 status = papiServiceCreate(&handle,
1560                             NULL,
1561                             NULL,
1562                             NULL,
1563                             NULL,
1564                             PAPI_ENCRYPT_NEVER,
1565                             NULL);
1566
1567 if (status != PAPI_OK)
1568 {
1569     /* handle the error */
1570     ...
1571 }
1572
1573 status = papiPrinterListJobs(handle,
1574                             printer_name,
1575                             job_attrs,
1576                             PAPI_LIST_JOBS_ALL,
1577                             0,
```

```

1577         &jobs);
1578     if (status != PAPI_OK)
1579     {
1580         /* handle the error */
1581         fprintf(stderr, "papiPrinterListJobs failed: %s\n",
1582             papiServiceGetStatusMessage(handle));
1583         ...
1584     }
1585     if (jobs != NULL)
1586     {
1587         for(i=0; jobs[i] != NULL; i++)
1588         {
1589             /* process the job */
1590             ...
1591         }
1592         papiJobListFree(jobs);
1593     }
1594     papiServiceDestroy(handle);
1595
1596
1597

```

1598

1599

See Also

1600

papiJobQuery, papiJobListFree

1601

5.8. papiPrinterGetAttributeList

1602

Description

1603

Get the attribute list associated with a printer object.

1604

Syntax

1605

1606

1607

1608

```

papi_attribute_t** papiPrinterGetAttributeList(
    papi_printer_t printer );

```

1609

1610

Inputs

1611

1612

printer

1613

Handle of the printer object.

1614

1615

Outputs

1616

none

1617

Returns

1618

Pointer to the attribute list associated with the printer object.

1619

Example

1620

1621

1622

1623

1624

1625

1626

1627

1628

1629

1630

```

#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,

```

```

1631         NULL,
1632         NULL,
1633         NULL,
1634         PAPI_ENCRYPT_NEVER,
1635         NULL);
1636     if (status != PAPI_OK)
1637     {
1638         /* handle the error */
1639         ...
1640     }
1641
1642     status = papiPrinterQuery(handle,
1643                             printer_name,
1644                             NULL,
1645                             &printer);
1646
1647     if (status != PAPI_OK)
1648     {
1649         /* handle the error */
1650         fprintf(stderr, "papiPrinterQuery failed: %s\n",
1651                papiServiceGetStatusMessage(handle));
1652         ...
1653     }
1654
1655     if (printer != NULL)
1656     {
1657         /* process the printer object */
1658         attrs = papiPrinterGetAttributeList(printer);
1659         ...
1660         papiPrinterFree(printer);
1661     }
1662     papiServiceDestroy(handle);
1663

```

1664

See Also

1665

1666 papiPrintersList, papiPrinterQuery

5.9. papiPrinterFree

1668

Description

1669

Free a printer object.

1670

Syntax

1671

```

1672 void papiPrinterFree(
1673         papi_printer_t    printer );
1674

```

1675

1676

Inputs

1677

1678 printer

1679

Handle of the printer object to free.

1680

1681

Outputs

1682

none

1683

Returns

1684

none

1685

Example

1686

```

1687 #include "papi.h"
1688
1689 papi_status_t status;
1690 papi_service_t handle = NULL;
1691 const char* printer_name = "my-printer";
1692 papi_printer_t printer = NULL;
1693 ...
1694 status = papiServiceCreate(&handle,
1695                             NULL,
1696                             NULL,
1697                             NULL,
1698                             NULL,
1699                             PAPI_ENCRYPT_NEVER,
1700                             NULL);
1701
1702 if (status != PAPI_OK)
1703 {
1704     /* handle the error */
1705     ...
1706 }
1707
1708 status = papiPrinterQuery(handle,
1709                             printer_name,
1710                             NULL,
1711                             &printer);
1712
1713 if (status != PAPI_OK)
1714 {
1715     /* handle the error */
1716     fprintf(stderr, "papiPrinterQuery failed: %s\n",
1717             papiServiceGetStatusMessage(handle));
1718     ...
1719 }
1720
1721 if (printer != NULL)
1722 {
1723     /* process the printer object */
1724     ...
1725     papiPrinterFree(printer);
1726 }
1727 papiServiceDestroy(handle);

```

1728

1729

See Also

1730

papiPrinterQuery

1731 **5.10. papiPrinterListFree**

1732

Description

1733

Free a list of printer objects.

1734

Syntax

1735

```

1736 void papiPrinterListFree(
1737     papi_printer_t* printers );
1738

```

1739

1740

Inputs

1741

1742 printers

1743

Pointer to the printer object list to free.

1744

1745 **Outputs**

1746 none

1747 **Returns**

1748 none

1749 **Example**

1750

```
1751           #include "papi.h"
1752
1753           papi_status_t status;
1754           papi_service_t handle = NULL;
1755           const char* printer_name = "my-printer";
1756           papi_printer_t* printers = NULL;
1757           ...
1758           status = papiServiceCreate(&handle,
1759                                      NULL,
1760                                      NULL,
1761                                      NULL,
1762                                      NULL,
1763                                      PAPI_ENCRYPT_NEVER,
1764                                      NULL);
1765
1766           if (status != PAPI_OK)
1767           {
1768           /* handle the error */
1769           ...
1770           }
1771
1772           status = papiPrinterList(handle,
1773                                    NULL,
1774                                    NULL,
1775                                    &printers);
1776
1777           if (status != PAPI_OK)
1778           {
1779           /* handle the error */
1780           fprintf(stderr, "papiPrinterList failed: %s\n",
1781                    papiServiceGetStatusMessage(handle));
1782           ...
1783           }
1784
1785           if (printers != NULL)
1786           {
1787           /* process the printer objects */
1788           ...
1789           papiPrinterListFree(printers);
1790           }
1791
1792           papiServiceDestroy(handle);
```

1792

1793 **See Also**

1794 papiPrinterList

1795 Chapter 6. Attributes API

1796 6.1. papiAttributeAdd

1797 Description

1798 Add an attribute/value to an attribute list. Depending on the `add_flags`, this may
1799 also be used to add values to an existing multivalued attribute. Memory is allocated
1800 and copies of the input arguments are created. It is the caller's responsibility to call
1801 `papiAttributeListFree` when done with the attribute list.

1802 This function is equivalent to the `papiAttributeAddString`,
1803 `papiAttributeAddInteger`, etc. functions defined later in this chapter.

1804 Syntax

1805

```
1806 papi_status_t papiAttributeAdd(  
1807     papi_attribute_t*** attrs,  
1808     const int add_flags,  
1809     const char* name,  
1810     const papi_attribute_value_type_t type,  
1811     const void* value );  
1812
```

1813

1814 Inputs

1815

1816 `attrs`

1817 Points to an attribute list. If a NULL value is passed, this function will allocate
1818 the attribute list.

1819 `add_flags`

1820 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
1821 that indicates how to handle the request.

1822 `name`

1823 Points to the name of the attribute to add.

1824 `type`

1825 The type of values for this attribute.

1826 `value`

1827 Points to the attribute value to be added.

1828

1829 Outputs

1830

1831 `attrs`

1832 The attribute list is updated.

1833

1834

Returns

1835

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

1836

1837

Example

1838

1839

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

1840

1841

1842

1843

1844

1845

1846

1847

1848

1849

1850

1851

1852

See Also

1853

papiAttributeListFree, papiAttributeAddString, papiAttributeAddInteger,

1854

papiAttributeAddBoolean, papiAttributeAddRange, papiAttributeAddResolution,

1855

papiAttributeAddDatetime

1856

6.2. papiAttributeAddString

1857

Description

1858

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.

1859

1860

1861

1862

Syntax

1863

1864

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

1865

1866

1867

1868

1869

1870

1871

Inputs

1872

1873 `attrs`

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

1874

1875

1876 `add_flags`

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

1877

1878

1879 name
 1880 Points to the name of the attribute to add.

1881 value
 1882 The value to be added.

1883

1884 **Outputs**

1885

1886 attrs
 1887 The attribute list is updated.

1888

1889 **Returns**

1890 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 1891 value is returned.

1892 **Example**

1893

```

1894 #include "papi.h"
1895
1896 papi_attribute_t** attrs = NULL;
1897 ...
1898 papiAttributeAddString(&attrs,
1899                       PAPI_EXCL,
1900                       "job-name",
1901                       "My job" );
1902 ...
1903 papiAttributeListFree(attrs);
1904
```

1905

1906 **See Also**

1907 papiAttributeListFree, papiAttributeAdd

1908 **6.3. papiAttributeAddInteger**

1909 **Description**

1910 Add an integer-valued attribute to an attribute list. Depending on the add_flags,
 1911 this may also be used to add values to an existing multivalued attribute. Memory is
 1912 allocated and copies of the input arguments are created. It is the caller's
 1913 responsibility to call papiAttributeListFree when done with the attribute list.

1914 **Syntax**

1915

```

1916 papi_status_t papiAttributeAddInteger(
1917     papi_attribute_t*** attrs,
1918     const int add_flags,
1919     const char* name,
1920     const int value );
1921
```

1922

1923 **Inputs**

1924

1925 attrs

1926 Points to an attribute list. If a NULL value is passed, this function will allocate
1927 the attribute list.

1928 add_flags

1929 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
1930 that indicates how to handle the request.

1931 name

1932 Points to the name of the attribute to add.

1933 value

1934 The value to be added.

1935

1936 **Outputs**

1937

1938 attrs

1939 The attribute list is updated.

1940

1941 **Returns**

1942 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1943 value is returned.

1944 **Example**

1945

```
1946           #include "papi.h"
1947
1948           papi_attribute_t** attrs = NULL;
1949           ...
1950           papiAttributeAddInteger(&attrs,
1951                                   PAPI_EXCL,
1952                                   "copies",
1953                                   3 );
1954           ...
1955           papiAttributeListFree(attrs);
1956
```

1957

1958 **See Also**

1959 papiAttributeListFree, papiAttributeAdd

1960 **6.4. papiAttributeAddBoolean**1961 **Description**

1962 Add a boolean-valued attribute to an attribute list. Depending on the add_flags,
1963 this may also be used to add values to an existing multivalued attribute. Memory is
1964 allocated and copies of the input arguments are created. It is the caller's
1965 responsibility to call papiAttributeListFree when done with the attribute list.

1966 **Syntax**

1967

```

1968     papi_status_t papiAttributeAddBoolean(
1969         papi_attribute_t*** attrs,
1970         const int add_flags,
1971         const char* name,
1972         const int value );
1973 
```

1974

1975 **Inputs**

1976

1977 attrs

1978 Points to an attribute list. If a NULL value is passed, this function will allocate
1979 the attribute list.

1980 add_flags

1981 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
1982 that indicates how to handle the request.

1983 name

1984 Points to the name of the attribute to add.

1985 value

1986 The value (0 or 1) to be added.

1987

1988 **Outputs**

1989

1990 attrs

1991 The attribute list is updated.

1992

1993 **Returns**

1994 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
1995 value is returned.

1996 **Example**

1997

```

1998     #include "papi.h"
1999
2000     papi_attribute_t** attrs = NULL;
2001     ...
2002     papiAttributeAddBoolean(&attrs,
2003         PAPI_EXCL,
2004         "color-supported",
2005         1 );
2006     ...
2007     papiAttributeListFree(attrs);
2008 
```

2009

2010 **See Also**

2011 `papiAttributeListFree`, `papiAttributeAdd`

2012 **6.5. papiAttributeAddRange**

2013 **Description**

2014 Add a range-valued attribute to an attribute list. Depending on the `add_flags`, this
 2015 may also be used to add values to an existing multivalued attribute. Memory is
 2016 allocated and copies of the input arguments are created. It is the caller's
 2017 responsibility to call `papiAttributeListFree` when done with the attribute list.

2018 **Syntax**

2019

```

2020 papi_status_t papiAttributeAddRange(
2021     papi_attribute_t*** attrs,
2022     const int add_flags,
2023     const char* name,
2024     const int lower,
2025     const int upper );
2026
    
```

2027

2028 **Inputs**

2029

2030 `attrs`

2031 Points to an attribute list. If a NULL value is passed, this function will allocate
 2032 the attribute list.

2033 `add_flags`

2034 A mask field consisting of one or more `PAPI_ATTR_*` values OR-ed together
 2035 that indicates how to handle the request.

2036 `name`

2037 Points to the name of the attribute to add.

2038 `lower`

2039 The lower range value.

2040 `upper`

2041 The upper range value.

2042

2043 **Outputs**

2044

2045 `attrs`

2046 The attribute list is updated.

2047

2048

Returns

2049

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

2050

2051

Example

2052

2053

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

2054

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

2055

```
...
```

2056

```
papiAttributeAddRange (&attrs,
                       PAPI_EXCL,
                       "job-k-octets-supported",
                       1,
                       100000 );
```

2057

```
...
```

2058

```
papiAttributeListFree (attrs);
```

2059

2060

2061

2062

2063

2064

2065

2066

See Also

2067

papiAttributeListFree

6.6. papiAttributeAddResolution

2068

Description

2069

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.

2070

2071

2072

2073

2074

Syntax

2075

2076

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

2077

2078

2079

2080

2081

2082

2083

2084

Inputs

2085

2086

attrs

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

2087

2088

2089

add_flags

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

2090

2091

2092

name

Points to the name of the attribute to add.

2093

2094 xres
2095 The X-axis resolution value.

2096 yres
2097 The Y-axis resolution value.

2098

2099 **Outputs**

2100

2101 attrs
2102 The attribute list is updated.

2103

2104 **Returns**

2105 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2106 value is returned.

2107 **Example**

2108

```
2109 #include "papi.h"  
2110  
2111 papi_attribute_t** attrs = NULL;  
2112 ...  
2113 papiAttributeAddResolution(&attrs,  
2114                            PAPI_EXCL,  
2115                            "printer-resolution",  
2116                            300,  
2117                            300 );  
2118 ...  
2119 papiAttributeListFree (attrs);  
2120
```

2121

2122 **See Also**

2123 papiAttributeListFree

2124 **6.7. papiAttributeAddDatetime**

2125 **Description**

2126 Add a date/time-valued attribute to an attribute list. Depending on the add_flags,
2127 this may also be used to add values to an existing multivalued attribute. Memory is
2128 allocated and copies of the input arguments are created. It is the caller's
2129 responsibility to call papiAttributeListFree when done with the attribute list.

2130 **Syntax**

2131

```
2132 papi_status_t papiAttributeAddDatetime(  
2133                   papi_attribute_t*** attrs,  
2134                   const int add_flags,  
2135                   const char* name,  
2136                   const time_t date_time );  
2137
```

2138

2139 **Inputs**

2140

2141 attrs

2142 Points to an attribute list. If a NULL value is passed, this function will allocate
2143 the attribute list.

2144 add_flags

2145 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together
2146 that indicates how to handle the request.

2147 name

2148 Points to the name of the attribute to add.

2149 date_time

2150 The date/time value.

2151

2152 **Outputs**

2153

2154 attrs

2155 The attribute list is updated.

2156

2157 **Returns**

2158 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
2159 value is returned.

2160 **Example**

2161

```

2162           #include "papi.h"
2163
2164           papi_attribute_t** attrs = NULL;
2165           time_t date_time
2166           ...
2167           time(&date_time);
2168           papiAttributeAddDatetime(&attrs,
2169                                    PAPI_EXCL,
2170                                    "date-time-at-creation",
2171                                    date_time );
2172           ...
2173           papiAttributeListFree(attrs);
2174

```

2175

2176 **See Also**

2177 papiAttributeListFree

2178 **6.8. papiAttributeListFree**2179 **Description**

2180 Frees an attribute list.

2181 **Syntax**

2182

```
2183           void papiAttributeListFree(  
2184                 const papi_attribute_t** attrs );  
2185
```

2186

2187 **Inputs**

2188

2189 attrs

 Attribute list to be freed.

2191

2192 **Outputs**

2193

 none

2194 **Returns**

2195

 none

2196 **Example**

2197

```
2198           #include "papi.h"  
2199  
2200           papi_attribute_t** attrs = NULL;  
2201           ...  
2202           papiAttributeAddString(&attrs,  
2203                                 "job-name",  
2204                                 PAPI_EXCL,  
2205                                 1,  
2206                                 "My job" );  
2207           ...  
2208           papiAttributeListFree(attrs);  
2209
```

2210

2211 **See Also**

2212

 papiAttributeAddString, etc.

2213 **6.9. papiAttributeListFind**

2214 **Description**

2215

 Find an attribute in an attribute list.

2216 **Syntax**

2217

```
2218           papi_attribute_t* papiAttributeListFind(  
2219                 const papi_attribute_t** attrs,  
2220                 const char*                 name );  
2221
```

2222

2223 **Inputs**

2224

2225 attrs
 2226 Attribute list to be searched.
 2227 name
 2228 Pointer to the name of the attribute to find.

2230 **Outputs**

2231 none

2232 **Returns**

2233 Pointer to the found attribute. NULL indicates that the specified attribute was not
 2234 found

2235 **Example**

2236

```

2237 #include "papi.h"
2238
2239 papi_attribute_t** attrs = NULL;
2240 papi_attribute_t* attr = NULL;
2241 ...
2242 attr = papiAttributeListFind(&attrs,
2243                             "job-name" );
2244
2245 if (attr != NULL)
2246 {
2247     /* process the attribute */
2248     ...
2249 }
2250 ...
2251 papiAttributeListFree (attrs);

```

2252

2253 **See Also**

2254 papiAttributeListGetNext

2255 **6.10. papiAttributeListGetNext**

2256 **Description**

2257 Get the next attribute in an attribute list.

2258 **Syntax**

2259

```

2260 papi_attribute_t* papiAttributeListGetNext(
2261     const papi_attribute_t** attrs,
2262     void** iterator );
2263

```

2264

2265 **Inputs**

2266

2267 attrs
 2268 Attribute list to be used.

2269 iterator

2270 Pointer to an opaque (void*) iterator. This should be NULL to find the first
2271 attribute and then passed in unchanged on subsequent calls to this function.

2272

2273 **Outputs**

2274 none

2275 **Returns**

2276 Pointer to the found attribute. NULL indicates that the end of the attribute list was
2277 reached.

2278 **Example**

2279

```
2280 #include "papi.h"  
2281  
2282 papi_attribute_t** attr = NULL;  
2283 papi_attribute_t* attr = NULL;  
2284 void* iterator = NULL;  
2285 ...  
2286 attr = papiAttributeListGetNext (&attr,  
2287                               &iterator );  
2288 while (attr != NULL)  
2289 {  
2290     /* process this attribute */  
2291     ...  
2292     attr = papiAttributeListGetNext (&attr,  
2293                                     &iterator );  
2294 }  
2295 ...  
2296 papiAttributeListFree (attr);  
2297
```

2298

2299 **See Also**

2300 papiAttributeListFind

2301 Chapter 7. Job API

2302 7.1. papiJobSubmit

2303 Description

2304 Submits a print job having the specified attributes to the specified printer.

2305 Syntax

2306

```
2307 papi_status_t papiJobSubmit(  
2308     papi_service_t handle,  
2309     const char* printer_name,  
2310     const papi_attribute_t** job_attributes,  
2311     const papi_job_ticket_t* job_ticket,  
2312     const char** file_names,  
2313     papi_job_t* job );  
2314
```

2315

2316 Inputs

2317

2318 handle

2319 Handle to the print service to use.

2320 printer_name

2321 Pointer to the name of the printer to which the job is to be submitted.

2322 job_attributes

2323 (optional) The list of attributes describing the job and how it is to be printed. If
2324 options are specified here and also in the job ticket data, the value specified
2325 here takes precedence. If this is NULL then only default attributes and
2326 (optionally) a job ticket is submitted with the job.

2327 job_ticket

2328 (optional) Pointer to structure specifying the job ticket. If this argument is
2329 NULL, then no job ticket is used with the job.

2330 file_names

2331 NULL terminated list of pointers to names of files to print.

2332

2333 Outputs

2334

2335 job

2336 The resulting job object representing the submitted job.

2337

2338

Returns

2339

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

2340

2341

Example

2342

2343

```

2344 #include "papi.h"
2345
2346 papi_status_t status;
2347 papi_service_t handle = NULL;
2348 const char* printer = "my-printer";
2349 const papi_attribute_t** attrs = NULL;
2350 const papi_job_ticket_t* ticket = NULL;
2351 const char* files[] = { "/etc/motd", NULL };
2352 papi_job_t job = NULL;
2353
2354 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
2355                           PAPI_ENCRYPT_IF_REQUESTED, NULL);
2356 if (status != PAPI_OK)
2357 {
2358     /* handle the error */
2359     ...
2360 }
2361
2362 papiAttributeAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
2363                       PAPI_STRING, 1, "test job");
2364 papiAttributeAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
2365                        PAPI_INTEGER, 1, 4);
2366
2367 status = papiJobSubmit(handle,
2368                       printer,
2369                       attrs,
2370                       ticket,
2371                       files,
2372                       &job);
2373 if (status != PAPI_OK)
2374 {
2375     fprintf(stderr, "papiJobSubmit failed: %s\n",
2376           papiStatusString(status));
2377     ...
2378 }
2379
2380 if (job != NULL)
2381 {
2382     /* look at the job object (maybe get the id) */
2383     papiJobFree(job);
2384 }
2385
2386 papiServiceDestroy(handle);
2387

```

2388

2389

See Also

2390

papiJobValidate, papiJobFree

2391

7.2. papiJobValidate

2392

Description

2393

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.

2394

2395

2396

Syntax

2397

2398

```

2399 papi_status_t papiJobValidate(
2400             papi_service_t handle,
2401             const char* printer_name,
2402             const papi_attribute_t** job_attributes,

```

```

2402         const papi_job_ticket_t*   job_ticket,
2403         const char**               file_names,
2404         papi_job_t*                 job );
2405

```

2406

2407 **Inputs**

2408

2409 handle

2410 Handle to the print service to use.

2411 printer_name

2412 Pointer to the name of the printer against which the job is to be validated.

2413 job_attributes

2414 (optional) The list of attributes describing the job and how it is to be printed. If
 2415 options are specified here and also in the job ticket data, the value specified
 2416 here takes precedence. If this is NULL then only default attributes and
 2417 (optionally) a job ticket is submitted with the job.

2418 job_ticket

2419 (optional) Pointer to structure specifying the JDF job ticket. If this argument is
 2420 NULL, then no job ticket is used with the job.

2421 file_names

2422 NULL terminated list of pointers to names of files to validate.

2423

2424 **Outputs**

2425

2426 job

2427 The resulting job object representing what would be the submitted job.

2428

2429 **Returns**

2430 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2431 value is returned.

2432 **Example**

2433

```

2434 #include "papi.h"
2435
2436 papi_status_t status;
2437 papi_service_t handle = NULL;
2438 const char* printer = "my-printer";
2439 const papi_attribute_t** attrs = NULL;
2440 const papi_job_ticket_t* ticket = NULL;
2441 const char* files[] = { "/etc/motd", NULL };
2442 papi_job_t job = NULL;
2443
2444 status = papiServiceCreate(&handle, NULL, NULL, NULL, NULL,
2445                          PAPI_ENCRYPT_IF_REQUESTED, NULL);
2446 if (status != PAPI_OK)
2447 {

```

```

2448         /* handle the error */
2449         ...
2450     }
2451
2452     papiAttributeAddString(&attrs, "job-name", PAPI_ATTR_EXCL,
2453                           PAPI_STRING, 1, "test job");
2454     papiAttributeAddInteger(&attrs, "copies", PAPI_ATTR_EXCL,
2455                            PAPI_INTEGER, 1, 4);
2456
2457     status = papiJobValidate(handle,
2458                             printer,
2459                             attrs,
2460                             ticket,
2461                             files,
2462                             &job);
2463
2464     if (status != PAPI_OK)
2465     {
2466         fprintf(stderr, "papiJobValidate failed: %s\n",
2467                papiStatusString(status));
2468         ...
2469     }
2470
2471     if (job != NULL)
2472     {
2473         ...
2474         papiJobFree(job);
2475     }
2476
2477     papiServiceDestroy(handle);

```

2478

2479

See Also

2480

papiJobSubmit, papiJobFree

2481

7.3. papiJobQuery

2482

Description

2483

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

2484

Syntax

2485

```

2486     papi_status_t papiJobQuery(
2487         papi_service_t    handle,
2488         const char*       printer_name,
2489         const int32_t     job_id,
2490         const char*       requested_attrs[],
2491         papi_job_t*      job );
2492

```

2493

2494

Inputs

2495

2496 handle

2497

Handle to the print service to use.

2498

printer_name

2499

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

2500

job_id

2501

The ID number of the job to be queried.

2502 requested_attrs

2503 NULL terminated array of attributes to be queried. If NULL is passed then all
 2504 available attributes are queried. (NOTE: The job may return more attributes
 2505 than you requested. This is merely an advisory request that may reduce the
 2506 amount of data returned if the printer/server supports it.)

2507

2508 **Outputs**

2509

2510 job

2511 The returned job object containing the requested attributes.

2512

2513 **Returns**

2514 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2515 value is returned.

2516 **Example**

2517

```

2518 #include "papi.h"
2519
2520 papi_status_t status;
2521 papi_service_t handle = NULL;
2522 const char* printer_name = "my-printer";
2523 papi_job_t job = NULL;
2524 int32_t job_id = 12;
2525 const char* job_attrs[] =
2526 {
2527     "job-id",
2528     "job-name",
2529     "job-originating-user-name",
2530     "job-state",
2531     "job-state-reasons",
2532     NULL
2533 };
2534 ...
2535 status = papiServiceCreate(&handle,
2536                             NULL,
2537                             NULL,
2538                             NULL,
2539                             NULL,
2540                             PAPI_ENCRYPT_NEVER,
2541                             NULL);
2542
2543 if (status != PAPI_OK)
2544 {
2545     /* handle the error */
2546     ...
2547 }
2548
2549 status = papiJobQuery(handle,
2550                       printer_name,
2551                       job_id,
2552                       job_attrs,
2553                       &job);
2554
2555 if (status != PAPI_OK)
2556 {
2557     /* handle the error */
2558     fprintf(stderr, "papiJobQuery failed: %s\n",
2559             papiServiceGetStatusMessage(handle));
2560     ...
2561 }
2562
2563 if (job != NULL)
2564 {
2565     /* process the job */
2566     ...
2567     papiJobFree(job);
2568 }
2569
2570 papiServiceDestroy(handle);

```

2570

2571

See Also

2572

papiJobFree, papiPrinterListJobs

2573

7.4. papiJobCancel

2574

Description

2575

Cancel the specified print job.

2576

Syntax

2577

2578

2579

2580

2581

2582

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

2583

2584

Inputs

2585

2586

handle

2587

Handle to the print service to use.

2588

printer_name

2589

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

2590

job_id

2591

The ID number of the job to be cancelled.

2592

2593

Outputs

2594

none

2595

Returns

2596

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

2597

value is returned.

2598

Example

2599

2600

2601

2602

2603

2604

2605

2606

2607

2608

2609

2610

2611

2612

2613

2614

2615

2616

```
#include "papi.h"

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

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



```

2617     ...
2618 }
2619
2620 status = papiJobCancel(handle,
2621                       printer_name,
2622                       job_id);
2623
2624 if (status != PAPI_OK)
2625 {
2626     /* handle the error */
2627     fprintf(stderr, "papiJobCancel failed: %s\n",
2628            papiServiceGetStatusMessage(handle));
2629     ...
2630 }
2631
2632 papiServiceDestroy(handle);

```

2633

2634 **See Also**

2635 papiPrinterListJobs, papiPrinterPurgeJobs

2636 **7.5. papiJobHold**2637 **Description**

2638 Holds the specified print job and prevents it from being scheduled for printing.
 2639 This operation is optional and may not be supported by all printers/servers. Use
 2640 papiJobRelease to undo the effects of this operation, or specify the hold_until
 2641 argument to automatically release the job at a specific time.

2642 **Syntax**

2643

```

2644 papi_status_t papiJobHold(
2645             papi_service_t      handle,
2646             const char*         printer_name,
2647             const int32_t       job_id,
2648             const char*         hold_until,
2649             const time_t*       hold_until_time );
2650

```

2651

2652 **Inputs**

2653

2654 handle

2655 Handle to the print service to use.

2656 printer_name

2657 Pointer to the name or URI of the printer to which the job was submitted.

2658 job_id

2659 The ID number of the job to be held.

2660 hold_until

2661 (optional) Specifies the time when the job will be automatically released for
 2662 printing. If NULL, the job is held until explicitly released by calling
 2663 papiJobRelease. If specified, the value must be one of the strings "indefinite"
 2664 (same effect as passing NULL), "day-time", "evening", "night", "weekend",
 2665 "second-shift", "third-shift", or "timed". For values other than "indefinite" and

2666 "timed", the printer/server must define exact times associated with these
 2667 values and it may make these associations configurable. If "timed" is specified,
 2668 then the hold_until_time argument is used.

2669 hold_until_time

2670 (optional) Specifies the time when the job will be automatically released for
 2671 printing. This argument is ignored unless "timed" is passed as the hold_until
 2672 argument.

2673

2674 **Outputs**

2675 none

2676 **Returns**

2677 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2678 value is returned.

2679 **Example**

2680

```

2681 #include "papi.h"
2682
2683 papi_status_t status;
2684 papi_service_t handle = NULL;
2685 const char* printer_name = "my-printer";
2686 int32_t job_id = 12;
2687 ...
2688 status = papiServiceCreate(&handle,
2689                          NULL,
2690                          NULL,
2691                          NULL,
2692                          NULL,
2693                          PAPI_ENCRYPT_NEVER,
2694                          NULL);
2695
2696 if (status != PAPI_OK)
2697 {
2698     /* handle the error */
2699     ...
2700 }
2701
2702 status = papiJobHold(handle,
2703                    printer_name,
2704                    job_id,
2705                    NULL,
2706                    NULL);
2707
2708 if (status != PAPI_OK)
2709 {
2710     /* handle the error */
2711     fprintf(stderr, "papiJobHold failed: %s\n",
2712            papiServiceGetStatusMessage(handle));
2713     ...
2714 }
2715 papiServiceDestroy(handle);
  
```

2716

2717 **See Also**

2718 papiJobRelease

2719 7.6. papiJobRelease

2720 **Description**

2721 Releases the specified print job, allowing it to be scheduled for printing. This
 2722 operation is optional and may not be supported by all printers/servers, but it must
 2723 be supported if papiJobHold is supported.

2724 **Syntax**

2725

```

2726     papi_status_t papiJobRelease(
2727         papi_service_t     handle,
2728         const char*        printer_name,
2729         const int32_t      job_id );
2730

```

2731

2732 **Inputs**

2733

2734 handle

2735 Handle to the print service to use.

2736 printer_name

2737 Pointer to the name or URI of the printer to which the job was submitted.

2738 job_id

2739 The ID number of the job to be released.

2740

2741 **Outputs**

2742 none

2743 **Returns**

2744 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
 2745 value is returned.

2746 **Example**

2747

```

2748     #include "papi.h"
2749
2750     papi_status_t status;
2751     papi_service_t handle = NULL;
2752     const char* printer_name = "my-printer";
2753     int32_t job_id = 12;
2754     ...
2755     status = papiServiceCreate(&handle,
2756         NULL,
2757         NULL,
2758         NULL,
2759         NULL,
2760         PAPI_ENCRYPT_NEVER,
2761         NULL);
2762
2763     if (status != PAPI_OK)
2764     {
2765         /* handle the error */
2766         ...
2767     }
2768
2769     status = papiJobRelease(handle,
2770         printer_name,
2771         job_id);
2772
2773     if (status != PAPI_OK)
2774     {
2775         /* handle the error */
2776         fprintf(stderr, "papiJobRelease failed: %s\n",
2777             papiServiceGetStatusMessage(handle));
2778         ...
2779     }
2780
2781     papiServiceDestroy(handle);

```

2780

2781

2782

See Also

2783

papiJobHold

2784

7.7. papiJobRestart

2785

Description

2786

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.

2787

2788

2789

Syntax

2790

2791

```
papi_status_t papiJobRestart(
2792             papi_service_t   handle,
2793             const char*      printer_name,
2794             const int32_t     job_id );
2795
```

2796

2797

Inputs

2798

2799

handle

2800

Handle to the print service to use.

2801

printer_name

2802

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

2803

job_id

2804

The ID number of the job to be restarted.

2805

2806

Outputs

2807

none

2808

Returns

2809

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

2810

2811

Example

2812

2813

```
#include "papi.h"
2814
2815 papi_status_t status;
2816 papi_service_t handle = NULL;
2817 const char* printer_name = "my-printer";
2818 int32_t job_id = 12;
2819 ...
2820 status = papiServiceCreate(&handle,
2821                          NULL,
2822                          NULL,
2823                          NULL,
```

```

2824
2825         NULL,
2826         PAPI_ENCRYPT_NEVER,
2827         NULL);
2828
2829     if (status != PAPI_OK)
2830     {
2831         /* handle the error */
2832         ...
2833     }
2834
2835     status = papiJobRestart(handle,
2836                           printer_name,
2837                           job_id);
2838
2839     if (status != PAPI_OK)
2840     {
2841         /* handle the error */
2842         fprintf(stderr, "papiJobRestart failed: %s\n",
2843                papiServiceGetStatusMessage(handle));
2844         ...
2845     }
2846
2847     papiServiceDestroy(handle);

```

2846

2847

See Also

2848

papiPrinterListJobs

2849 **7.8. papiJobGetAttributeList**

2850

Description

2851

Get the attribute list associated with a job object.

2852

Syntax

2853

```

2854 papi_attribute_t** papiJobGetAttributeList(
2855     papi_job_t      job );
2856

```

2857

2858

Inputs

2859

2860 job

2861

Handle of the job object.

2862

2863

Outputs

2864

none

2865

Returns

2866

Pointer to the attribute list associated with the job object.

2867

Example

2868

```

2869 #include "papi.h"
2870
2871 papi_status_t status;
2872 papi_service_t handle = NULL;
2873 const char* printer_name = "my-printer";
2874 papi_job_t job = NULL;
2875 papi_attribute_list* attrs = NULL;
2876 ...
2877 status = papiServiceCreate(&handle,

```

```

2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912

```

```

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

    status = papiJobQuery(handle,
                        printer_name,
                        67,
                        NULL,
                        &job);
    if (status != PAPI_OK)
    {
        /* handle the error */
        fprintf(stderr, "papiJobQuery failed: %s\n",
                papiServiceGetStatusMessage(handle));
        ...
    }

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

    papiServiceDestroy(handle);

```

2913

2914

See Also

2915

papiPrinterListJobs, papiJobQuery

2916 **7.9. papiJobGetPrinterName**

2917

Description

2918

Get the printer name associated with a job object.

2919

Syntax

2920

```

2921 char* papiJobGetPrinterName(
2922         papi_job_t    job );
2923

```

2924

2925

Inputs

2926

2927 job

Handle of the job object.

2929

2930

Outputs

2931

none

2932

Returns

2933

Pointer to the printer name associated with the job object.

2934

Example

2935

2936

```

#include "papi.h"

char* printer_name = NULL;
papi_job_t job = NULL;
...
if (job != NULL)
{
    /* process the job object */
    printer_name = papiJobGetPrinterName(job);
    ...
    papiJobFree(job);
}

```

2937

2938

2939

2940

2941

2942

2943

2944

2945

2946

2947

2948

2949

2950

See Also

2951

papiPrinterListJobs, papiJobQuery

2952

7.10. papiJobGetId

2953

Description

2954

Get the job ID associated with a job object.

2955

Syntax

2956

2957

```

int32_t papiJobGetId(
        papi_job_t    job );

```

2958

2959

2960

2961

Inputs

2962

2963 job

Handle of the job object.

2964

2965

2966

Outputs

2967

none

2968

Returns

2969

The job ID associated with the job object.

2970

Example

2971

2972

```

#include "papi.h"

int32_t job_id;
papi_job_t job = NULL;
...
if (job != NULL)
{
    /* process the job object */
    job_id = papiJobGetId(job);
    ...
    papiJobFree(job);
}

```

2973

2974

2975

2976

2977

2978

2979

2980

2981

2982

2983

2984

2985

2986 **See Also**

2987 papiPrinterListJobs, papiJobQuery

2988 7.11. papiJobGetJobTicket

2989 **Description**

2990 Get the job ticket associated with a job object.

2991 **Syntax**

2992

```
2993 papi_job_ticket_t* papiJobGetJobTicket(  
2994     papi_job_t job );  
2995
```

2996

2997 **Inputs**

2998

2999 job

Handle of the job object.

3001

3002 **Outputs**

3003 none

3004 **Returns**

3005 Pointer to the job ticket associated with the job object.

3006 **Example**

3007

```
3008 #include "papi.h"  
3009  
3010 papi_job_ticket_t* job_ticket = NULL;  
3011 papi_job_t job = NULL;  
3012 ...  
3013 if (job != NULL)  
3014 {  
3015     /* process the job object */  
3016     job_ticket = papiJobGetJobTicket(job);  
3017     ...  
3018     papiJobFree(job);  
3019 }  
3020
```

3021

3022 **See Also**

3023 papiPrinterListJobs, papiJobQuery

3024 7.12. papiJobFree

3025 **Description**

3026 Free a job object.

3027

Syntax

3028

3029

```
void papiJobFree(
3030         papi_job_t    job );
3031
```

3032

3033

Inputs

3034

3035 job

Handle of the job object to free.

3037

3038

Outputs

3039

none

3040

Returns

3041

none

3042

Example

3043

3044

```
#include "papi.h"
3045
3046 papi_status_t status;
3047 papi_service_t handle = NULL;
3048 const char* printer_name = "my-printer";
3049 papi_job_t job = NULL;
3050 ...
3051 status = papiServiceCreate(&handle,
3052                             NULL,
3053                             NULL,
3054                             NULL,
3055                             NULL,
3056                             PAPI_ENCRYPT_NEVER,
3057                             NULL);
3058
3059 if (status != PAPI_OK)
3060 {
3061     /* handle the error */
3062     ...
3063 }
3064
3065 status = papiJobQuery(handle,
3066                       printer_name,
3067                       12,
3068                       &job);
3069
3070 if (status != PAPI_OK)
3071 {
3072     /* handle the error */
3073     fprintf(stderr, "papiJobQuery failed: %s\n",
3074             papiServiceGetStatusMessage(handle));
3075     ...
3076 }
3077
3078 if (job != NULL)
3079 {
3080     /* process the job object */
3081     ...
3082     papiJobFree(job);
3083 }
3084
3085 papiServiceDestroy(handle);
```

3085

3086 **See Also**

3087 papiJobQuery

3088 **7.13. papiJobListFree**

3089 **Description**

3090 Free a list of job objects.

3091 **Syntax**

3092

```
3093           void papiJobListFree(
3094                                papi_job_t*        jobs );
3095
```

3096

3097 **Inputs**

3098

3099 jobs

 Pointer to the job object list to free.

3101

3102 **Outputs**

3103 none

3104 **Returns**

3105 none

3106 **Example**

3107

```
3108           #include "papi.h"
3109
3110           papi_status_t status;
3111           papi_service_t handle = NULL;
3112           const char* printer_name = "my-printer";
3113           papi_job_t* jobs = NULL;
3114           ...
3115           status = papiServiceCreate(&handle,
3116                                    NULL,
3117                                    NULL,
3118                                    NULL,
3119                                    NULL,
3120                                    PAPI_ENCRYPT_NEVER,
3121                                    NULL);
3122
3123           if (status != PAPI_OK)
3124           {
3125               /* handle the error */
3126               ...
3127           }
3128
3129           status = papiPrinterListJobs(handle,
3130                                        printer_name,
3131                                        NULL,
3132                                        0, 0, 0,
3133                                        &jobs);
3134
3135           if (status != PAPI_OK)
3136           {
3137               /* handle the error */
3138               fprintf(stderr, "papiPrinterListJobs failed: %s\n",
3139                        papiServiceGetStatusMessage(handle));
3140               ...
3141           }
3142           if (jobs != NULL)
3143           {
```

```
3143     /* process the job objects */  
3144     ...  
3145     papiJobListFree(jobs);  
3146 }  
3147  
3148 papiServiceDestroy(handle);  
3149
```

3150

3151

See Also

3152

papiPrinterListJobs

3153 Chapter 8. Miscellaneous API

3154 8.1. papiStatusString

3155 Description

3156 Get a status string for the specified papi_status_t. The status message returned
3157 from this function may be less detailed than the status message returned from
3158 papiServiceGetStatusMessage (if the print service supports returning more detailed
3159 error messages).

3160 The returned message will be localized in the language of the submitter of the
3161 requestor.

3162 Syntax

3163

```
3164 char* papiStatusString(  
3165     const papi_status_t status );  
3166
```

3167

3168 Inputs

3169

3170 status

3171 The status value to convert to a status string.

3172

3173 Outputs

3174 none

3175 Returns

3176 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure
3177 value is returned.

3178 Example

3179

```
3180 #include "papi.h"  
3181 papi_status_t status;  
3182 ...  
3183 fprintf(stderr, "PAPI function failed: %s\n", papiStatusString(status));  
3184  
3185
```

3186

3187 See Also

3188 papiServiceGetStatusMessage

3189 **Chapter 9. Attributes**

3190 * *ISSUE: Waiting for reference to single document from Tom H.*
3191

3192 **9.1. Extension Attributes**

3193 The following attributes are not currently defined by IPP, but may be used with
3194 this API.

3195 **9.1.1. job-ticket-formats-supported**

3196 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats
3197 that are supported by the printer. If this attribute is not present, it is assumed that
3198 the printer does not support any job ticket formats.

3199 * *ISSUE: I took the following required attr lists directly from IPP RFC 2911 to use as a starting point. We probably
3200 want to add/delete attrs from the lists.*
3201

3202 **9.2. Required Job Attributes**

3203 The following job attributes *must* be supported to comply with this API standard.
3204 These attributes may be supported by the underlying print server directly, or they
3205 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

3206

3207 **9.3. Required Printer Attributes**

3208 The following printer attributes *must* be supported to comply with this API
3209 standard. These attributes may be supported by the underlying print server
3210 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

3211 printer-name
printer-state
printer-state-reasons
printer-up-time
printer-uri-supported
queued-job-count
uri-authentication-supported
uri-security-supported

3212 **Appendix A. Change History**

3213 **Version 0.4 (July 19, 2002)**

3214

3215 • Made papi_job_t and papi_printer_t opaque handles and added "get" functions
3216 to access the associated information (papiPrinterGetAttributeList,
3217 papiJobGetAttributeList, papiJobGetId, papiJobGetPrinterName,
3218 papiJobGetJobTicket).

3219 • Removed variable length argument lists from attribute add functions.

3220 • Changed order and name of flag value passed to attribute add functions.

3221 • Eliminated indirection in date/time value passed to papiAttributeAddDatetime.

3222 • Added message argument to papiPrinterPause.

3223

3224 **Version 0.3 (June 24, 2002)**

3225

3226 • Converted to DocBook format from Microsoft Word

3227 • Major rewrite, including:

3228 • Changed how printer names are described in "Model/Printer"

3229 • Changed fixed length strings to pointers in numerous structures/sections

3230 • Redefined attribute/value structures and associated API descriptions

3231 • Changed list/query functions to return "objects"

3232 • Rewrote "Attributes API" chapter

3233 • Changed many function definitions to pass NULL-terminated arrays of
3234 pointers instead of a separate count argument

3235 • Changed papiJobSubmit to take an attribute list structure as input instead of a
3236 formatted string

3237

3238

3239 **Version 0.2 (April 17, 2002)**

3240

3241 • Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)

3242 • Filled in "Encryption" section and added information about encryption in "Object
3243 Identification" section

3244 • Added "short_name" field in "Object Identification" section

3245 • Added "Job Ticket (papi_job_ticket_t)" section

3246 • Added papiPrinterPause

3247 • Added papiPrinterResume

3248 • Added papiPurgeJobs

3249 • Added optional job_ticket argument to papiJobSubmit

Appendix A. Change History

- 3250 • Added optional passing of filenames by URI to papiJobSubmit
- 3251 • Added papiHoldJob
- 3252 • Added papiReleaseJob
- 3253 • Added papiRestartJob

3254

Version 0.1 (April 3, 2002)

3256

- 3257 • Original draft version

3258

3259

3260

3261

3262

3263

<i>End of Document</i>
