

1

2

Open Standard Print API (PAPI)

3

Version 1.0

4

Alan Hlava
IBM Printing Systems Division

5

6

7

Norm Jacobs
Sun Microsystems, Inc.

8

9

10

Michael R. Sweet
Easy Software Products

11

12 **Open Standard Print API (PAPI): Version 1.0**

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

14

15 Version 1.0 Edition

16 Copyright © 2002-2005 Free Standards Group

17

18 Permission to use, copy, modify and distribute this document for any purpose and without

19 fee is hereby granted in perpetuity, provided that the above copyright notice and this

20 paragraph appear in all copies.

21

Table of Contents

Chapter 1: Introduction.....	7
Chapter 2: Print System Model.....	8
2.1 Introduction.....	8
2.2 Model.....	8
2.3 Security.....	9
2.4 Globalization.....	10
Chapter 3: Common Structures.....	11
3.1 Conventions.....	11
3.2 Service Object (papi_service_t).....	11
3.3 Attributes and Values (papi_attribute_t).....	11
3.4 Job Object (papi_job_t).....	13
3.5 Stream Object (papi_stream_t).....	13
3.6 Printer Object (papi_printer_t).....	13
3.7 Job Ticket (papi_job_ticket_t).....	13
3.8 Status (papi_status_t).....	14
3.9 List Filter (papi_filter_t).....	15
3.10 Encryption (papi_encrypt_t).....	17
Chapter 4: Attributes API.....	18
4.1 papiAttributeListAddValue.....	19
4.2 papiAttributeListAddString.....	20
4.3 papiAttributeListAddInteger.....	21
4.4 papiAttributeListAddBoolean.....	22
4.5 papiAttributeListAddRange.....	24
4.6 papiAttributeListAddResolution.....	25
4.7 papiAttributeListAddDatetime.....	26
4.8 papiAttributeListAddCollection.....	28
4.9 papiAttributeListAddMetadata.....	29
4.10 papiAttributeListDelete.....	30
4.11 papiAttributeListGetValue.....	31
4.12 papiAttributeListGetString.....	33
4.13 papiAttributeListGetInteger.....	34
4.14 papiAttributeListGetBoolean.....	36
4.15 papiAttributeListGetRange.....	37
4.16 papiAttributeListGetResolution.....	38
4.17 papiAttributeListGetDatetime.....	40
4.18 papiAttributeListGetCollection.....	41
4.19 papiAttributeListGetMetadata.....	43
4.20 papiAttributeListFree.....	44
4.21 papiAttributeListFind.....	45
4.22 papiAttributeListGetNext.....	46
4.23 papiAttributeListFromString.....	47
4.24 papiAttributeListToString.....	48

Chapter 5: Service API.....	50
5.1 papiServiceCreate.....	50
5.2 papiServiceDestroy.....	52
5.3 papiServiceSetUserName.....	52
5.4 papiServiceSetPassword.....	54
5.5 papiServiceSetEncryption.....	55
5.6 papiServiceSetAuthCB.....	56
5.7 papiServiceSetAppData.....	57
5.8 papiServiceGetServiceName.....	58
5.9 papiServiceGetUserName.....	59
5.10 papiServiceGetPassword.....	59
5.11 papiServiceGetEncryption.....	60
5.12 papiServiceGetAppData.....	61
5.13 papiServiceGetAttributeList.....	62
5.14 papiServiceGetStatusMessage.....	63
Chapter 6: Printer API.....	65
6.1 papiPrintersList.....	65
6.2 papiPrinterQuery.....	67
6.3 papiPrinterModify.....	69
6.4 papiPrinterAdd.....	70
6.5 papiPrinterRemove.....	72
6.6 papiPrinterPause.....	73
6.7 papiPrinterResume.....	74
6.8 papiPrinterEnable.....	75
6.9 papiPrinterDisable.....	76
6.10 papiPrinterPurgeJobs.....	77
6.11 papiPrinterListJobs.....	79
6.12 papiPrinterGetAttributeList.....	81
6.13 papiPrinterFree.....	82
6.14 papiPrinterListFree.....	82
Chapter 7: Job API.....	84
7.1 papiJobSubmit.....	84
7.2 papiJobSubmitByReference.....	86
7.3 papiJobValidate.....	88
7.4 papiJobStreamOpen.....	90
7.5 papiJobStreamWrite.....	92
7.6 papiJobStreamClose.....	93
7.7 papiJobQuery.....	94
7.8 papiJobModify.....	96
7.9 papiJobCancel.....	97
7.10 papiJobHold.....	98
7.11 papiJobRelease.....	99
7.12 papiJobRestart.....	100
7.13 papiJobPromote.....	101

7.14 papiJobGetAttributeList.....	103
7.15 papiJobGetPrinterName.....	104
7.16 papiJobGetId.....	104
7.17 papiJobGetJobTicket.....	105
7.18 papiJobFree.....	106
7.19 papiJobListFree.....	107
Chapter 8: Miscellaneous API.....	108
8.1 papiStatusString.....	108
8.2 papiLibrarySupportedCalls.....	108
8.3 papiLibrarySupportedCall.....	109
Chapter 9: Capabilities.....	111
9.1 Introduction.....	111
9.2 Objectives.....	112
9.3 Interfaces.....	113
Chapter 10: Attributes.....	117
10.1 Extension Attributes.....	117
10.2 Required Job Attributes.....	117
10.3 Required Printer Attributes.....	118
10.4 IPP Attribute Type Mapping.....	118
Chapter 11: Attribute List Text Representation.....	120
11.1 ABNF Definition.....	120
11.2 Examples.....	121
Chapter 12: Conformance.....	123
12.1 Query Profile.....	123
12.2 Job Submission Profile.....	123
12.3 Conformance Table.....	123
Chapter 13: Sample Code.....	127
Chapter 14: References.....	128
14.1 Internet Printing Protocol (IPP).....	128
14.2 Job Ticket.....	128
14.3 Printer Working Group (PWG).....	128
14.4 Other.....	128
Chapter 15: Change History.....	129
15.1 Version 1.0 (July 14, 2005).....	129
15.2 Version 1.0 (May 9, 2005).....	129
15.3 Version 0.92 (January 12, 2005).....	129
15.4 Version 0.91 (January 28, 2004).....	129
15.5 Version 0.9 (November 18, 2002).....	130
15.6 Version 0.8 (November 15, 2002).....	130
15.7 Version 0.7 (October 18, 2002).....	130
15.8 Version 0.6 (September 20, 2002).....	130
15.9 Version 0.5 (August 30, 2002).....	131
15.10 Version 0.4 (July 19, 2002).....	132
15.11 Version 0.3 (June 24, 2002).....	132

22

15.12 Version 0.2 (April 17, 2002).....	132
15.13 Version 0.1 (April 3, 2002).....	133

23 **Chapter 1: Introduction**

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

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

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

36 Subsequent versions of this document will incorporate support for a Document object,
37 notification, and additional functions and attributes to more completely align with the [PWG](#)
38 [semantic model](#).

39 Chapter 2: Print System Model

40 2.1 Introduction

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

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

- 46 • Object Definition: What objects are part of the model?
- 47 • Object Naming: How is each object identified/named?
- 48 • Object Relationships: What are the associations and relationships between the
49 objects?

50 Some possible objects a printing system model might include are:

Printer	Queue	Print Resources (font, etc.)
Document	Filter/Transform	Job Ticket
Medium/Form	Job	Auxiliary Sheet
Server	Class/Pool	

51

52 2.2 Model

53 The model on which the Open Standard Print API is derived from reflect the semantics
54 defined by the Internet Printing Protocol (IPP) standard. This is a fairly simple model in
55 terms of the number of object types. It is defined very clearly and in detail in the IPP
56 [RFC2911], Chapter 2. Additional IPP-related documents can be found in the [References](#)
57 appendix

58 Consult the above document for a thorough understanding of the IPP print model. A brief
59 summary of the model is provided here.

60 2.2.1 Print Service

61 Note that an implementation of the PAPI interface may use protocols other than IPP for
62 communicating with a print service. The only requirement is that the implementation
63 accept and return the data structures as defined in this document.

64 2.2.2 Printer

65 Printer objects are the target of print job requests. A printer object may represent an actual
66 printer (if the printer itself supports PAPI), an object in a server representing an actual
67 printer, or an abstract object in a server (perhaps representing a pool or class of printers).

68 Printer objects are identified by one or more names which may be short, local names (such

69 as "prtr1") or longer global names (such as a URI like
70 "<http://printserv.mycompany.com:631/printers/prtr1>", "ipp://printserv/printers/prt1",
71 "lpd://server/queue", etc.). The PAPI implementation may detect and map short names to
72 long global names in an implementation-specific manner.

73 **2.2.3 Job**

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

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

80 **2.2.4 Document**

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

84 This version of PAPI does NOT support separate document objects, but they will be added
85 in a future version. It is likely that this will be done by adding new "Open job", "Add
86 document", and "Close job" functions to allow submitting a multiple document job and
87 specifying separate attributes for each document.

88 **2.3 Security**

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

91 **2.3.1 Authentication**

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

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

98 **2.3.2 Authorization**

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

101 Since authorization is an entirely server-side (or printer-side) function, how it works is not
102 specified by this API. In other words, the server (or printer) may or may not do

103 authorization checking according to its capability and current configuration. If
104 authorization checking is performed, any call may fail with an error code indicating the
105 failure (PAPI_NOT_AUTHORIZED).

106 **2.3.3 Encryption**

107 Encrypting certain data sent to and from the print service may be desirable in some
108 environments. See the "encryption" field in the service object for information on how to
109 request encryption on a print operation. Note that some print services may not support
110 encryption. To comply with this standard, only the PAPI_ENCRYPT_NEVER value must
111 be supported.

112 **2.4 Globalization**

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

- 115 • Attribute names are never translated.
- 116 • Most text values are not translated.
- 117 • Supporting translation by PAPI implementation is optional.
- 118 • If translation is supported, only the values of the following attributes are
119 translated: job-state-message, document-state-message, and printer-state-
120 message.

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

123 Chapter 3: Common Structures

124 3.1 Conventions

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

130 3.2 Service Object (*papi_service_t*)

131 This opaque structure is used as a "handle" to maintain information about the print service
132 being used to handle the PAPI requests. It is typically created once, used on one or more
133 subsequent PAPI calls, and then destroyed.

```
134 typedef void *papi_service_t;
```

135

136 Included in the information associated with a *papi_service_t* is a definition about how
137 requests will be encrypted during communication with the print service.

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

144

145 Note that to comply with this standard, only the *PAPI_ENCRYPT_NEVER* value must be
146 supported.

147 3.3 Attributes and Values (*papi_attribute_t*)

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

```
149 /* Attribute Type */  
150 typedef enum {  
151     PAPI_STRING,  
152     PAPI_INTEGER,  
153     PAPI_BOOLEAN,  
154     PAPI_RANGE,  
155     PAPI_RESOLUTION,  
156     PAPI_DATETIME,  
157     PAPI_COLLECTION
```

Chapter 3: Common Structures

```
158     PAPI_METADATA
159 } papi_attribute_value_type_t;
160
161 /* Resolution units */
162 typedef enum {
163     PAPI_RES_PER_INCH = 3,
164     PAPI_RES_PER_CM
165 } papi_res_t; /* Boolean values */
166
167 enum {
168     PAPI_FALSE = 0,
169     PAPI_TRUE = 1
170 };
171
172 typedef enum {
173     PAPI_UNSUPPORTED = 0x10,
174     PAPI_DEFAULT = 0x11,
175     PAPI_UNKNOWN,
176     PAPI_NO_VALUE,
177     PAPI_NOT_SETTABLE = 0x15,
178     PAPI_DELETE = 0x16
179 } papi_metadata_t;
180
181 struct papi_attribute_str;
182
183 /* Attribute Value */
184 typedef union {
185     char *string; /* PAPI_STRING value */
186     int integer; /* PAPI_INTEGER value */
187     char boolean; /* PAPI_BOOLEAN value */
188     struct { /* PAPI_RANGE value */
189         int lower;
190         int upper;
191     } range;
192     struct { /* PAPI_RESOLUTION value */
193         int xres;
194         int yres;
195         papi_res_t units;
196     } resolution
197     time_t datetime; /* PAPI_DATETIME value */
198     struct papi_attribute_str **
199         collection; /* PAPI_COLLECTION value */
200     papi_metadata_t metadata;
```

```

201 } papi_attribute_value_t;
202
203 /* Attribute and Values */
204 typedef struct papi_attribute_str {
205     char *name; /* attribute name */
206     papi_attribute_value_type_t type; /* type of values */
207     papi_attribute_value_t **values; /* list of values */
208 } papi_attribute_t;

```

209

210 The following constants are used by the papiAttributeListAdd* functions to control how
211 values are added to the list.

```

212 /* Attribute add flags (add_flags) */
213 #define PAPI_ATTR_APPEND    0x0001 /* Add values to attribute*/
214 #define PAPI_ATTR_REPLACE  0x0002 /* Delete existing values, then add */
215 #define PAPI_ATTR_EXCL     0x0004 /* Fail if attribute exists */

```

216

217 For the valid attribute names which may be supported, see The [Attributes](#) appendix.

218 **3.4 Job Object (papi_job_t)**

219 This opaque structure is used as a "handle" to information associated with a job object. This
220 handle is returned in response to successful job creation, modification, query, or list
221 operations. See the "papiJobGet*" functions to see what information can be retrieved from
222 the job object using the handle.

223 **3.5 Stream Object (papi_stream_t)**

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

226 **3.6 Printer Object (papi_printer_t)**

227 This opaque structure is used as a "handle" to information associated with a printer object.
228 This handle is returned in response to successful printer modification, query, or list
229 operations. See the "papiPrinterGet*" functions to see what information can be retrieved
230 from the printer object using the handle.

231 **3.7 Job Ticket (papi_job_ticket_t)**

232 This structure is used to pass a job ticket when submitting a print job. Currently, Job
233 Definition Format (JDF) is the only supported job ticket format. JDF is an XML- based job
234 ticket syntax. The JDF specification can be found at <http://www.cip4.org/>.

Chapter 3: Common Structures

```
235 /* Job Ticket Format */
236 typedef enum {
237     PAPI_JT_FORMAT_JDF = 0,      /* Job Definition Format */
238     PAPI_JT_FORMAT_PWG = 1      /* PWG Job Ticket Format */
239 } papi_jt_format_t;
240
241 /* Job Ticket */
242 typedef struct papi_job_ticket_s {
243     papi_jt_format_t format;      /* Format of job ticket */
244     char *ticket_data;           /* Buffer containing the job ticket data. If NULL,
245                                  file_name must be specified */
246     char *file_name;            /* Name of the file containing the job ticket data.
247                                  If ticket_data is specified, then file_name
248                                  is ignored. */
249 } papi_job_ticket_t;
```

250

251 The file_name field may contain absolute path names, relative path names or URIs
252 ([RFC1738], [RFC2396]). In the event that the name contains an absolute or relative path
253 name (relative to the current directory), the implementation MUST copy the file contents
254 before returning. If the name contains a URI, the implementation SHOULD NOT copy the
255 referenced data unless (or until) it is no longer feasible to maintain the reference. Feasibility
256 limitations may arise out of security issues, name space issues, and/or protocol or printer
257 limitations.

258 **3.8 Status (*papi_status_t*)**

```
259 typedef enum {
260     PAPI_OK = 0x0000,
261     PAPI_OK_SUBST,
262     PAPI_OK_CONFLICT,
263     PAPI_OK_IGNORED_SUBSCRIPTIONS,
264     PAPI_OK_IGNORED_NOTIFICATIONS,
265     PAPI_OK_TOO_MANY_EVENTS,
266     PAPI_OK_BUT_CANCEL_SUBSCRIPTION,
267     PAPI_REDIRECTION_OTHER_SITE = 0x300,
268     PAPI_BAD_REQUEST = 0x0400,
269     PAPI_FORBIDDEN,
270     PAPI_NOT_AUTHENTICATED,
271     PAPI_NOT_AUTHORIZED,
272     PAPI_NOT_POSSIBLE,
273     PAPI_TIMEOUT,
274     PAPI_NOT_FOUND,
275     PAPI_GONE,
```

```

276 PAPI_REQUEST_ENTITY,
277 PAPI_REQUEST_VALUE,
278 PAPI_DOCUMENT_FORMAT,
279 PAPI_ATTRIBUTES,
280 PAPI_URI_SCHEME,
281 PAPI_CHARSET,
282 PAPI_CONFLICT,
283 PAPI_COMPRESSION_NOT_SUPPORTED,
284 PAPI_COMPRESSION_ERROR,
285 PAPI_DOCUMENT_FORMAT_ERROR,
286 PAPI_DOCUMENT_ACCESS_ERROR,
287 PAPI_ATTRIBUTES_NOT_SETTABLE,
288 PAPI_IGNORED_ALL_SUBSCRIPTIONS,
289 PAPI_TOO_MANY_SUBSCRIPTIONS,
290 PAPI_IGNORED_ALL_NOTIFICATIONS,
291 PAPI_PRINT_SUPPORT_FILE_NOT_FOUND,
292 PAPI_INTERNAL_ERROR = 0x0500,
293 PAPI_OPERATION_NOT_SUPPORTED,
294 PAPI_SERVICE_UNAVAILABLE,
295 PAPI_VERSION_NOT_SUPPORTED,
296 PAPI_DEVICE_ERROR,
297 PAPI_TEMPORARY_ERROR,
298 PAPI_NOT_ACCEPTING,
299 PAPI_PRINTER_BUSY,
300 PAPI_ERROR_JOB_CANCELLED,
301 PAPI_MULTIPLE_JOBS_NOT_SUPPORTED,
302 PAPI_PRINTER_IS_DEACTIVATED,
303 PAPI_BAD_ARGUMENT,
304 PAPI_JOB_TICKET_NOT_SUPPORTED
305 } papi_status_t;

```

306

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

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

311 **3.9 List Filter (*papi_filter_t*)**

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

315 typedef enum {

Chapter 3: Common Structures

```
316     PAPI_FILTER_BITMASK = 0
317     /* future filter types may be added here */
318 } papi_filter_type_t;
319
320 typedef struct {
321     papi_filter_type_t type; /* Type of filter specified */
322     union {
323         /* Bitmask filter */
324         struct {
325             unsigned int mask; /* bit mask */
326             unsigned int value; /* bit value */
327         } bitmask;
328     /* future filter types may be added here */
329     } filter;
330 } papi_filter_t;
```

331

332 For [papiPrintersList](#) requests, the following values may be OR-ed together and used in the
333 `papi_filter_t` mask and value fields to limit the printers returned. The logic used is to select
334 printers which satisfy: "(printer-type & mask) == (value & mask)". This allows for simple
335 "positive logic" (checking for the presence of characteristics) when mask and value are
336 identical, and it also allows for "negative logic" (checking for the absence of characteristics)
337 when they are different. For example, to select local (i.e. NOT remote) printers that support
338 color:

```
339 papi_filter_t filter; filter.type = PAPI_FILTER_BITMASK;
340 filter.filter.bitmask.mask = PAPI_PRINTER_REMOTE | PAPI_PRINTER_COLOR;
341 filter.filter.bitmask.value = PAPI_PRINTER_COLOR;
```

342 The filter bitmask values are:

```
343 enum {
344     PAPI_PRINTER_LOCAL = 0x0000, /* Local printer or class */
345     PAPI_PRINTER_CLASS = 0x0001, /* Printer class */
346     PAPI_PRINTER_REMOTE = 0x0002, /* Remote printer or class */
347     PAPI_PRINTER_BW = 0x0004, /* Can do B&W printing */
348     PAPI_PRINTER_COLOR = 0x0008, /* Can do color printing */
349     PAPI_PRINTER_DUPLEX = 0x0010, /* Can do duplexing */
350     PAPI_PRINTER_STAPLE = 0x0020, /* Can staple output */
351     PAPI_PRINTER_COPIES = 0x0040, /* Can do copies */
352     PAPI_PRINTER_COLLATE = 0x0080, /* Can collage copies */
353     PAPI_PRINTER_PUNCH = 0x0100, /* Can punch output */
354     PAPI_PRINTER_COVER = 0x0200, /* Can cover output */
355     PAPI_PRINTER_BIND = 0x0400, /* Can bind output */
356     PAPI_PRINTER_SORT = 0x0800, /* Can sort output */
```



```
357     PAPI_PRINTER_SMALL = 0x1000,      /* Can do Letter/Legal/A4 */
358     PAPI_PRINTER_MEDIUM = 0x2000,    /* Can do Tabloid/B/C/A3/A2 */
359     PAPI_PRINTER_LARGE = 0x4000,     /* Can do D/E/A1/A0 */
360     PAPI_PRINTER_VARIABLE = 0x8000,  /* Can do variable sizes */
361     PAPI_PRINTER_IMPLICIT = 0x10000, /* Implicit class */
362     PAPI_PRINTER_DEFAULT = 0x20000,  /* Default printer on network */
363     PAPI_PRINTER_OPTIONS = 0xfffc    /* ~(CLASS | REMOTE | IMPLICIT) */
364 };
```

365 **3.10 Encryption (*papi_encrypt_t*)**

366 This enumeration is used to get/set the encryption type to be used during communication
367 with the print service.

```
368 typedef enum {
369     PAPI_ENCRYPT_IF_REQUESTED,
370     PAPI_ENCRYPT_NEVER,
371     PAPI_ENCRYPT_REQUIRED,
372     PAPI_ENCRYPT_ALWAYS
373 } papi_encryption_t;
```

374

375 **Chapter 4: Attributes API**

376 The interface described in this section is central to the PAPI printing model. Virtually all of
377 the operations that can be performed against the print service objects (via function calls)
378 make use of attributes. Object creation or modification operations tend to take in attribute
379 list describing the object or the requested modifications. Object creation, modification,
380 query and list operations tend to return updated lists of print service objects containing
381 attribute lists to more completely describe the objects.

382 In the case of a printer object, its associated attribute list can be retrieved using
383 [papiPrinterGetAttributeList](#). Job object attribute lists can be retrieved using
384 [papiJobGetAttributeList](#). Once retrieved, these attribute lists can be searched (or
385 enumerated) to gather further information about the associated object. When creating or
386 modifying print service objects, attribute lists can be built and passed into the create/modify
387 operation. As a general rule of thumb, application developers should not modify or destroy
388 attribute lists that they did not create. Modification or destruction of attribute lists retrieved
389 from print service objects should be handled by the PAPI implementation upon object
390 destruction (free).

391 Because the attribute interface has specific functions to ease the use of various types of data
392 that can be contained in an attribute list, there are a few things that are common to all of the
393 `papiAttributeAdd*` functions and some common to all of the `papiAttributeListGet*`
394 functions.

395 All of the `papiAttributeListAdd*` functions take in a pointer to an attribute list, a set of
396 flags, an attribute name, and call/type specific values. For all of the `papiAttributeListAdd*`
397 functions, the attribute list pointer (`papi_attribute_t ***attrs`) may not contain a NULL
398 value. If a NULL value is passed to any of these functions, the function must return
399 `PAPI_BAD_ARGUMENT`. The flags passed into each of the `papiAttributeListAdd*` calls
400 describe how the attribute/values are to be added to the attribute list. Currently, there are
401 three flags that can be passed: `PAPI_ATTR_EXCL`, `PAPI_ATTR_REPLACE`, and
402 `PAPI_ATTR_APPEND`. If `PAPI_ATTR_EXCL` is passed, it indicates that this call should
403 only succeed if the named attribute does not already exist in the attribute list.
404 `PAPI_ATTR_REPLACE` indicates that prior to addition to the attribute list, this call should
405 truncate any existing attribute values for the named attribute if it is already contained in the
406 list. `PAPI_ATTR_APPEND` indicates that any attribute values contained in this call should
407 be appended to the named attribute's value list if the named attribute was already contained
408 in the attribute list.

409 All of the `papiAttributeListGet*` functions take in an attribute list, iterator, name, and
410 pointer(s) for type specific results. If the named attribute is found in the attribute list, but
411 its type does not match the type supplied in `papiAttributeListGet` or the type implied by the
412 various type specific calls, a value of `PAPI_NOT_POSSIBLE` must be returned from the
413 call. Any `papiAttributeListGet*` failure must not modify the information in the provided
414 results arguments.

415 **4.1 *papiAttributeListAddValue***

416 **4.1.1 Description**

417 Add an attribute/value to an attribute list. Depending on the `add_flags`, this may also be
418 used to add values to an existing multi-valued attribute. Memory is allocated and copies of
419 the input arguments are created. It is the caller's responsibility to call [papiAttributeListFree](#)
420 when done with the attribute list.

421 This function is equivalent to the [papiAttributeListAddString](#), [papiAttributeListAddInteger](#),
422 [papiAttributeListAddBoolean](#), [papiAttributeListAddRange](#)
423 [papiAttributeListAddResolution](#), [papiAttributeListAddDatetime](#),
424 [papiAttributeListAddCollection](#), and [papiAttributeListAddMetadata](#) functions defined later
425 in this chapter.

426 **4.1.2 Syntax**

```
427 papi_status_t papiAttributeListAddValue(papi_attribute_t ***attrs, int add_flags,  
428                                     char *name, papi_attribute_value_type_t type,  
429                                     papi_attribute_value_t *value );
```

430 **4.1.3 Inputs**

431 **4.1.3.1 *attrs***

432 Points to an attribute list. If `*attrs` is NULL then this function will allocate the attribute list.

433 **4.1.3.2 *add_flags***

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

436 **4.1.3.3 *name***

437 Points to the name of the attribute to add.

438 **4.1.3.4 *type***

439 The type of values for this attribute.

440 **4.1.3.5 *value***

441 Points to the attribute value to be added.

442 **4.1.4 Outputs**

443 **4.1.4.1 *attrs***

444 The attribute list is updated.

445 **4.1.5 Returns**

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

448 **4.1.6 Example**

```
449 papi_status_t status;  
450 papi_attribute_t **attrs = NULL;  
451 papi_attribute_value_t value;  
452 ...  
453 value.string = "My Job";  
454 status = papiAttributeListAddValue(&attrs, PAPI_ATTR_EXCL, "job-name",  
455                                     PAPI_STRING, &value);  
456 ...  
457 papiAttributeListFree(attrs);
```

458 **4.1.7 See Also**

459 [papiAttributeListAddString](#), [papiAttributeListAddInteger](#), [papiAttributeListAddBoolean](#),
460 [papiAttributeListAddRange](#), [papiAttributeListAddResolution](#),
461 [papiAttributeListAddDatetime](#), [papiAttributeListAddCollection](#)
462 [papiAttributeListFromString](#), [papiAttributeListFree](#)

463 **4.2 *papiAttributeListAddString***

464 **4.2.1 Description**

465 Add a string-valued attribute to an attribute list. Depending on the add_flags, this may also
466 be used to add values to an existing multi-valued attribute. Memory is allocated and copies
467 of the input arguments are created. It is the caller's responsibility to call
468 [papiAttributeListFree](#) when done with the attribute list.

469 **4.2.2 Syntax**

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

472 **4.2.3 Inputs**

473 **4.2.3.1 *attrs***

474 Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

475 **4.2.3.2 *add_flags***

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

477 indicates how to handle the request.

478 **4.2.3.3 name**

479 Points to the name of the attribute to add.

480 **4.2.3.4 value**

481 The string value to be added to the attribute.

482 **4.2.4 Outputs**

483 **4.2.4.1 attrs**

484 The attribute list is updated.

485 **4.2.5 Returns**

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

488 **4.2.6 Example**

```
489 papi_status_t status;  
490 papi_attribute_t **attrs = NULL;  
491 ...  
492 status = papiAttributeListAddString(&attrs, PAPI_ATTR_EXCL,  
493                                     "job-name", "My job" );  
494 ...  
495 papiAttributeListFree(attrs);
```

496 **4.2.7 See Also**

497 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

498 **4.3 papiAttributeListAddInteger**

499 **4.3.1 Description**

500 Add an integer-valued attribute to an attribute list. Depending on the `add_flags`, this may
501 also be used to add values to an existing multi-valued attribute. Memory is allocated and
502 copies of the input arguments are created. It is the caller's responsibility to call
503 [papiAttributeListFree](#) when done with the attribute list.

504 **4.3.2 Syntax**

```
505 papi_status_t papiAttributeListAddInteger(papi_attribute_t ***attrs, int add_flags,  
506                                           char *name,int value );
```

507 **4.3.3 Inputs**

508 **4.3.3.1 *attrs***

509 Points to an attribute list. If **attrs* is NULL then this function will allocate the attribute list.

510 **4.3.3.2 *add_flags***

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

513 **4.3.3.3 *name***

514 Points to the name of the attribute to add.

515 **4.3.3.4 *value***

516 The integer value to be added to the attribute.

517 **4.3.4 Outputs**

518 **4.3.4.1 *attrs***

519 The attribute list is updated.

520 **4.3.5 Returns**

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

523 **4.3.6 Example**

```
524 papi_status_t status;  
525 papi_attribute_t **attrs = NULL;  
526 ...  
527 status = papiAttributeListAddInteger(&attrs, PAPI_ATTR_EXCL,  
528                                     "copies", 3);  
529 ...  
530 papiAttributeListFree(attrs);
```

531 **4.3.7 See Also**

532 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

533 **4.4 *papiAttributeListAddBoolean***

534 **4.4.1 Description**

535 Add a boolean-valued attribute to an attribute list. Depending on the *add_flags*, this may

536 also be used to add values to an existing multi-valued attribute. Memory is allocated and
537 copies of the input arguments are created. It is the caller's responsibility to call
538 [papiAttributeListFree](#) when done with the attribute list.

539 **4.4.2 Syntax**

```
540 papi_status_t papiAttributeListAddBoolean(papi_attribute_t ***attrs, int add_flags,  
541 char *name, char value );
```

542 **4.4.3 Inputs**

543 **4.4.3.1 attrs**

544 Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

545 **4.4.3.2 add_flags**

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

548 **4.4.3.3 name**

549 Points to the name of the attribute to add.

550 **4.4.3.4 value**

551 The boolean value (PAPI_FALSE or PAPI_TRUE) to be added to the attribute.

552 **4.4.4 Outputs**

553 **4.4.4.1 attrs**

554 The attribute list is updated.

555 **4.4.5 Returns**

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

558 **4.4.6 Example**

```
559 papi_status_t status;  
560 papi_attribute_t **attrs = NULL;  
561 ...  
562 status = papiAttributeListAddBoolean(&attrs, PAPI_ATTR_EXCL,  
563                                     "color-supported", PAPI_TRUE);  
564 ...  
565 papiAttributeListFree(attrs);
```

566 **4.4.7 See Also**

567 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

568 **4.5 papiAttributeListAddRange**

569 **4.5.1 Description**

570 Add a range-valued attribute to an attribute list. Depending on the `add_flags`, this may also
571 be used to add values to an existing multi-valued attribute. Memory is allocated and copies
572 of the input arguments are created. It is the caller's responsibility to call
573 [papiAttributeListFree](#) when done with the attribute list.

574 **4.5.2 Syntax**

```
575 papi_status_t papiAttributeListAddRange(papi_attribute_t ***attrs, int add_flags,  
576                                         char *name, int lower, int upper);
```

577 **4.5.3 Inputs**

578 **4.5.3.1 attrs**

579 Points to an attribute list. If `*attrs` is NULL then this function will allocate the attribute list.

580 **4.5.3.2 add_flags**

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

583 **4.5.3.3 name**

584 Points to the name of the attribute to add.

585 **4.5.3.4 lower**

586 An integer value representing the lower boundary of a range value. This value must be less
587 than or equal to the upper range value.

588 **4.5.3.5 upper**

589 An integer value representing the upper boundary of the range value. This value must be
590 greater than or equal to the lower range value

591 **4.5.4 Outputs**

592 **4.5.4.1 attrs**

593 The attribute list is updated.

594 **4.5.5 Returns**

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

597 **4.5.6 Example**

```
598 papi_status_t status;  
599 papi_attribute_t **attrs = NULL;  
600 ...  
601 status = papiAttributeListAddRange(&attrs, PAPI_ATTR_EXCL,  
602                                     "job-k-octets-supported",1, 100000);  
603 ...  
604 papiAttributeListFree(attrs);
```

605 **4.5.7 See Also**

606 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

607 **4.6 papiAttributeListAddResolution**

608 **4.6.1 Description**

609 Add a resolution-valued attribute to an attribute list. Depending on the add_flags, this may
610 also be used to add values to an existing multi-valued attribute. Memory is allocated and
611 copies of the input arguments are created. It is the caller's responsibility to call
612 [papiAttributeListFree](#) when done with the attribute list.

613 **4.6.2 Syntax**

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

617 **4.6.3 Inputs**

618 **4.6.3.1 attrs**

619 Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

620 **4.6.3.2 add_flags**

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

623 **4.6.3.3 name**

624 Points to the name of the attribute to add.

625 **4.6.3.4 xres**

626 The integer X-axis resolution value.

627 **4.6.3.5 yres**

628 The integer Y-axis resolution value.

629 **4.6.3.6 Units**

630 The units of the X-axis and y-axis resolution values provided.

631 **4.6.4 Outputs**

632 **4.6.4.1 attrs**

633 The attribute list is updated.

634 **4.6.5 Returns**

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

637 **4.6.6 Example**

```
638 papi_status_t status;  
639 papi_attribute_t **attrs = NULL;  
640 ...  
641 status = papiAttributeListAddResolution(&attrs, PAPI_ATTR_EXCL,  
642                                         "printer-resolution", 300, 300,  
643                                         PAPI_RES_PER_INCH);  
644 ...  
645 papiAttributeListFree(attrs);
```

646 **4.6.7 See Also**

647 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

648 **4.7 papiAttributeListAddDatetime**

649 **4.7.1 Description**

650 Add a date/time-valued attribute to an attribute list. Depending on the add_flags, this may
651 also be used to add values to an existing multi-valued attribute. Memory is allocated and
652 copies of the input arguments are created. It is the caller's responsibility to call
653 [papiAttributeListFree](#) when done with the attribute list.

654 **4.7.2 Syntax**

```
655 papi_status_t papiAttributeListAddDatetime(papi_attribute_t ***attrs, int add_flags,  
656 char *name, time_t value );
```

657 **4.7.3 Inputs**

658 **4.7.3.1 attrs**

659 Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

660 **4.7.3.2 add_flags**

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

663 **4.7.3.3 name**

664 Points to the name of the attribute to add.

665 **4.7.3.4 value**

666 The time_t representation of the date/time value to be added to the attribute.

667 **4.7.4 Outputs**

668 **4.7.4.1 attrs**

669 The attribute list is updated.

670 **4.7.5 Returns**

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

673 **4.7.6 Example**

```
674 papi_status_t status;  
675 papi_attribute_t ***attrs = NULL;  
676 time_t date_time;  
677 ...  
678 time(&date_time);  
679 status = papiAttributeListAddValue(&attrs, PAPI_EXCL,  
680 "date-time-at-creation", date_time);  
681 ...  
682 papiAttributeListFree(attrs);
```

683 **4.7.7 See Also**

684 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

685 **4.8 papiAttributeListAddCollection**

686 **4.8.1 Description**

687 Add a collection-valued attribute to an attribute list. A collection-valued attribute is a
688 container for list of attributes. Depending on the `add_flags`, this may also be used to add
689 values to an existing multi-valued attribute. Memory is allocated and copies of the input
690 arguments are created. It is the caller's responsibility to call [papiAttributeListFree](#) when
691 done with the attribute list.

692 **4.8.2 Syntax**

```
693 papi_status_t papiAttributeListAddCollection(papi_attribute_t ***attrs,  
694                                             int add_flags, char *name,  
695                                             papi_attribute_t **collection);
```

696 **4.8.3 Inputs**

697 **4.8.3.1 attrs**

698 Points to an attribute list. If `*attrs` is NULL then this function will allocate the attribute list.

699 **4.8.3.2 add_flags**

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

702 **4.8.3.3 name**

703 Points to the name of the attribute to add.

704 **4.8.3.4 collection**

705 Points to the attribute list to be added as a collection.

706 **4.8.4 Outputs**

707 **4.8.4.1 attrs**

708 The attribute list is updated.

709 **4.8.5 Returns**

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

711 returned.

712 **4.8.6 Example**

```
713 papi_status_t status;
714 papi_attribute_t **attrs = NULL;
715 papi_attribute_t **collection = NULL;
716 ...
717 /* create the collection /
718 status = papiAttributeListAddString(&collection, PAPI_EXCL,
719                                     "media-key", "iso-a4-white");
720 status = papiAttributeListAddString(&collection, PAPI_EXCL,
721                                     "media-type", "stationery");
722 ...
723 / add the collection to the attribute list */
724 status = papiAttributeListAddCollection(&attrs, PAPI_EXCL,
725                                         "media-col", collection);
726 ...
727 papiAttributeListFree(collection);
728 papiAttributeListFree(attrs);
```

729 **4.8.7 See Also**

730 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

731 **4.9 papiAttributeListAddMetadata**

732 **4.9.1 Description**

733 Add a meta-valued attribute to an attribute list. A meta-valued attribute is a container for
734 attribute information not normally represented in an attribute value. Memory is allocated
735 and copies of the input arguments are created. It is the caller's responsibility to call
736 [papiAttributeListFree](#) when done with the attribute list.

737 **4.9.2 Syntax**

```
738 papi_status_t papiAttributeListAddMetadata(papi_attribute_t ***attrs,
739                                           int add_flags, char *name,
740                                           papi_metadata_t value);
```

741 **4.9.3 Inputs**

742 **4.9.3.1 attrs**

743 Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

744 **4.9.3.2 *add_flags***

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

747 **4.9.3.3 *name***

748 Points to the name of the attribute to add.

749 **4.9.3.4 *value***

750 The type of metadata to be added to the attribute. PAPI_DELETE can be used to indicate
751 that an attribute should be removed from a print service object when calling one of the
752 papi*Modify functions. PAPI_DEFAULT can be used to indicate that the print service
753 should set (or reset) the named attribute value to a “default” value during a create or modify
754 operation of a print service object.

755 **4.9.4 Outputs**

756 **4.9.4.1 *attrs***

757 The attribute list is updated.

758 **4.9.5 Returns**

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

761 **4.9.6 Example**

```
762 papi_status_t status;  
763 papi_attribute_t **attrs = NULL;  
764 ...  
765 / add the collection to the attribute list */  
766 status = papiAttributeListAddMetadata(&attrs, PAPI_EXCL,  
767                                     "media", PAPI_DELETE);  
768 ...  
769 papiAttributeListFree(collection);  
770 papiAttributeListFree(attrs);
```

771 **4.9.7 See Also**

772 [papiAttributeListAddValue](#), [papiAttributeListFree](#)

773 **4.10 *papiAttributeListDelete***

774 **4.10.1 Description**

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

776 deallocated.

777 **4.10.2 Syntax**

```
778 papi_status_t papiAttributeListDelete(papi_attribute_t ***attrs, char *name);
```

779 **4.10.3 Inputs**

780 **4.10.3.1 attrs**

781 Points to an attribute list.

782 **4.10.3.2 name**

783 Points to the name of the attribute to remove.

784 **4.10.4 Outputs**

785 **4.10.4.1 attrs**

786 The attribute list is updated.

787 **4.10.5 Returns**

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

790 **4.10.6 Example**

```
791 papi_status_t status;  
792 papi_attribute_t **attrs = NULL;  
793 ...  
794 status = papiAttributeListDelete(&attrs, "copies");  
795 ...  
796 papiAttributeListFree(attrs);
```

797 **4.10.7 See Also**

798 [papiAttributeListFree](#)

799 **4.11 papiAttributeListGetValue**

800 **4.11.1 Description**

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

802 This function is equivalent to the [papiAttributeListGetString](#), [papiAttributeListGetInteger](#),
803 [papiAttributeListGetBoolean](#), [papiAttributeListGetRange](#), [papiAttributeListGetResolution](#),
804 [papiAttributeListGetDatetime](#), and [papiAttributeListGetCollection](#) functions defined later in

805 this chapter.

806 **4.11.2 Syntax**

```
807 papi_status_t papiAttributeListGetValue(papi_attribute_t **attrs, void **iterator,  
808                                         char *name, papi_attribute_value_type_t type,  
809                                         papi_attribute_t **value);
```

810 **4.11.3 Inputs**

811 **4.11.3.1 *attrs***

812 Points to an attribute list.

813 **4.11.3.2 *iterator***

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

818 **4.11.3.3 *name***

819 Points to the name of the attribute to retrieve. If the named attribute can not be located in
820 the attribute list supplied, PAPI_NOT_FOUND is returned.

821 **4.11.3.4 *type***

822 The type of values for this attribute. If the type supplied does not match the type of the
823 named attribute in the attribute list, PAPI_NOT_POSSIBLE is returned.

824 **4.11.4 Outputs**

825 **4.11.4.1 *iterator***

826 See [iterator](#) in the [Inputs](#) section above

827 **4.11.4.2 *value***

828 Points to the variable where a pointer to the attribute value is to be returned. Note that the
829 returned pointer points to the attribute's value in the list (no copy of the value is made) so
830 that the caller does not need to do any special cleanup of the returned value's memory (it is
831 cleaned up when the containing attribute list is deallocated).

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

833 **4.11.5 Returns**

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

835 returned.

836 **4.11.6 Example**

```
837 papi_status_t status;  
838 papi_attribute_t **attrs = NULL;  
839 papi_attribute_value_t *job_name_value;  
840 ...  
841 status = papiAttributeListGetValue(attrs, NULL, "job-name",  
842                                     PAPI_STRING, &job_name_value);  
843 ...  
844 papiAttributeListFree(attrs);
```

845 **4.11.7 See Also**

846 [papiAttributeListGetString](#), [papiAttributeListGetInteger](#), [papiAttributeListGetBoolean](#),
847 [papiAttributeListGetRange](#), [papiAttributeListGetResolution](#), [papiAttributeListGetDatetime](#),
848 [papiAttributeListGetCollection](#), [papiAttributeListFree](#)

849 **4.12 papiAttributeListGetString**

850 **4.12.1 Description**

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

852 **4.12.2 Syntax**

```
853 papi_status_t papiAttributeListGetString(papi_attribute_t **attrs, void **iterator,  
854                                         char *name, char **value);
```

855 **4.12.3 Inputs**

856 **4.12.3.1 *attrs***

857 Points to an attribute list.

858 **4.12.3.2 *iterator***

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

863 **4.12.3.3 *name***

864 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
865 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
866 found in the attribute list, but is not a PAPI_STRING, PAPI_NOT_POSSIBLE will be

867 returned.

868 **4.12.4 Outputs**

869 **4.12.4.1 Iterator**

870 See [iterator](#) in the [Inputs](#) section above

871 **4.12.4.2 value**

872 Pointer to the string (char *) where a pointer to the value is returned. If this call returns an
873 error, the output value is not changed. Note that the returned pointer points to the attribute's
874 value in the list (no copy of the value is made) so that the caller does not need to perform
875 any special cleanup of the returned value's memory (it is cleaned up when the containing
876 attribute list is deallocated).

877 **4.12.5 Returns**

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

880 **4.12.6 Example**

```
881 papi_status_t status;  
882 papi_attribute_t **attrs = NULL;  
883 char *value = NULL;  
884 ...  
885 status = papiAttributeListGetString(attrs, NULL, "job-name",  
886                                     PAPI_STRING, &value);  
887 ...  
888 papiAttributeListFree(attrs);
```

889 **4.12.7 See Also**

890 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

891 **4.13 papiAttributeListGetInteger**

892 **4.13.1 Description**

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

894 **4.13.2 Syntax**

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

897 **4.13.3 Inputs**

898 **4.13.3.1 *attrs***

899 Points to an attribute list.

900 **4.13.3.2 *iterator***

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

905 **4.13.3.3 *name***

906 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
907 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
908 found in the attribute list, but is not a PAPI_INTEGER, PAPI_NOT_POSSIBLE will be
909 returned.

910 **4.13.4 Outputs**

911 **4.13.4.1 *iterator***

912 See [iterator](#) in the [Inputs](#) section above

913 **4.13.4.2 *value***

914 Pointer to the int where the value is returned. The value from the attribute list is copied to
915 this location. If this call returns an error, the output value is not changed.

916 **4.13.5 Returns**

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

919 **4.13.6 Example**

```
920 papi_status_t status;  
921 papi_attribute_t **attrs = NULL;  
922 int value = 0;  
923 ...  
924 status = papiAttributeListGetInteger(attrs, NULL, "copies", &value);  
925 ...  
926 papiAttributeListFree(attrs);
```

927 **4.13.7 See Also**

928 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

929 **4.14 *papiAttributeListGetBoolean***

930 **4.14.1 Description**

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

932 **4.14.2 Syntax**

```
933 papi_status_t papiAttributeListGetBoolean(papi_attribute_t **attrs, void **iterator,  
934 char *name, char *value);
```

935 **4.14.3 Inputs**

936 **4.14.3.1 *attrs***

937 Points to an attribute list.

938 **4.14.3.2 *iterator***

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

943 **4.14.3.3 *name***

944 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
945 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
946 found in the attribute list, but is not a PAPI_BOOLEAN, PAPI_NOT_POSSIBLE will be
947 returned.

948 **4.14.4 Outputs**

949 **4.14.4.1 *iterator***

950 See [iterator](#) in the [Inputs](#) section above.

951 **4.14.4.2 *value***

952 Pointer to the char where the value is returned. The value from the attribute list is copied to
953 this location. If this call returns an error, the output value is not changed.

954 **4.14.5 Returns**

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

957 **4.14.6 Example**

```
958 papi_status_t status;  
959 papi_attribute_t **attrs = NULL;  
960 char value = PAPI_FALSE;  
961 ...  
962 status = papiAttributeListGetBoolean(attrs, NULL,  
963                                     "color-supported", &value);  
964 ...  
965 papiAttributeListFree(attrs);
```

966 **4.14.7 See Also**

967 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

968 **4.15 papiAttributeListGetRange**

969 **4.15.1 Description**

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

971 **4.15.2 Syntax**

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

974 **4.15.3 Inputs**

975 **4.15.3.1 attrs**

976 Points to an attribute list.

977 **4.15.3.2 iterator**

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

982 **4.15.3.3 name**

983 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
984 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
985 found in the attribute list, but is not a PAPI_RANGE, PAPI_NOT_POSSIBLE will be
986 returned.

987 **4.15.4 Outputs**

988 **4.15.4.1 Iterator**

989 See [iterator](#) in the [inputs](#) section above.

990 **4.15.4.2 lower**

991 Pointer to the integer where the values are returned. The value from the attribute list is
992 copied to this location. If this call returns an error, the output values are not changed.

993 **4.15.4.3 upper**

994 Pointer to the integer where the values are returned. The value from the attribute list is
995 copied to this location. If this call returns an error, the output values are not changed.

996 **4.15.5 Returns**

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

999 **4.15.6 Example**

```
1000 papi_status_t status;  
1001 papi_attribute_t **attrs = NULL;  
1002 int lower = 0;  
1003 int upper = 0;  
1004 ...  
1005 status = papiAttributeListGetRange(attrs, NULL,  
1006                                     "job-k-octets-supported", &lower, &upper);  
1007 ...  
1008 papiAttributeListFree(attrs);
```

1009 **4.15.7 See Also**

1010 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1011 **4.16 papiAttributeListGetResolution**

1012 **4.16.1 Description**

1013 Get a resolution-valued attribute's value from an attribute list.

1014 **4.16.2 Syntax**

```
1015 papi_status_t papiAttributeListGetResolution(papi_attribute_t **attrs, void **iterator,  
1016                                             char *name, int *xres, int *yres, papi_res_t *units);
```

1017 **4.16.3 Inputs**

1018 **4.16.3.1 *attrs***

1019 Points to an attribute list.

1020 **4.16.3.2 *iterator***

1021 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1022 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1023 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1024 be passed in unchanged on subsequent calls to this function to get the remaining values.

1025 **4.16.3.3 *name***

1026 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1027 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1028 found in the attribute list, but is not a PAPI_RESOLUTION, PAPI_NOT_POSSIBLE will
1029 be returned.

1030 **4.16.4 Outputs**

1031 **4.16.4.1 *iterator***

1032 See [iterator](#) in the [Inputs](#) section above.

1033 **4.16.4.2 *xres***

1034 Pointer to the int where the X-resolution value is returned. The value from the attribute list
1035 is copied to this location. If this call returns an error, the output value is not changed.

1036 **4.16.4.3 *yres***

1037 Pointer to the int where the Y-resolution value is returned. The value from the attribute list
1038 is copied to this location. If this call returns an error, the output value is not changed.

1039 **4.16.4.4 *units***

1040 Pointer to the variable where the resolution-units value is returned. The value from the
1041 attribute list is copied to this location. If this call returns an error, the output value is not
1042 changed.

1043 **4.16.5 Returns**

1044 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1045 returned.

1046 **4.16.6 Example**

```
1047 papi_status_t status;  
1048 papi_attribute_t **attrs = NULL;  
1049 int xres, yres;  
1050 papi_res_t units;  
1051 ...  
1052 status = papiAttributeListGetResolution(attrs, NULL,  
1053                                     "printer-resolution", &xres, &yres, &units);  
1054 ...  
1055 papiAttributeListFree(attrs);
```

1056 **4.16.7 See Also**

1057 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1058 **4.17 papiAttributeListGetDatetime**

1059 **4.17.1 Description**

1060 Get a datetime-valued attribute's value from an attribute list.

1061 **4.17.2 Syntax**

```
1062 papi_status_t papiAttributeListGetDatetime(papi_attribute_t **attrs, void **iterator,  
1063                                           char *name, time_t *value);
```

1064 **4.17.3 Inputs**

1065 **4.17.3.1 attrs**

1066 Points to an attribute list.

1067 **4.17.3.2 iterator**

1068 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1069 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1070 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1071 be passed in unchanged on subsequent calls to this function to get the remaining values.

1072 **4.17.3.3 name**

1073 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1074 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1075 found in the attribute list, but is not a PAPI_DATETIME, PAPI_NOT_POSSIBLE will be
1076 returned.

1077 **4.17.4 Outputs**

1078 **4.17.4.1 iterator**

1079 See [iterator](#) in the [Inputs](#) section above

1080 **4.17.4.2 value**

1081 Pointer to the `time_t` where the value is returned. The value from the attribute list is copied
1082 to this location. If this call returns an error, the output value is not changed.

1083 **4.17.5 Returns**

1084 If successful, a value of `PAPI_OK` is returned. Otherwise an appropriate failure value is
1085 returned.

1086 **4.17.6 Example**

```
1087 papi_status_t status;  
1088 papi_attribute_t **attrs = NULL;  
1089 time_t value = 0;  
1090 ...  
1091 status = papiAttributeListGetDatetime(attrs, NULL,  
1092                                     "date-time-at-creation", &value);  
1093 ...  
1094 papiAttributeListFree(attrs);
```

1095 **4.17.7 See Also**

1096 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1097 **4.18 papiAttributeListGetCollection**

1098 **4.18.1 Description**

1099 Get a collection-valued attribute's value from an attribute list.

1100 **4.18.2 Syntax**

```
1101 papi_status_t papiAttributeListGetCollection(papi_attribute_t **attrs, void **iterator,  
1102                                             char *name, papi_attribute_t ***value);
```

1103 **4.18.3 Inputs**

1104 **4.18.3.1 attrs**

1105 Points to an attribute list.

1106 **4.18.3.2 iterator**

1107 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1108 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1109 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1110 be passed in unchanged on subsequent calls to this function to get the remaining values.

1111 **4.18.3.3 name**

1112 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1113 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1114 found in the attribute list, but is not a PAPI_COLLECTION, PAPI_NOT_POSSIBLE will
1115 be returned.

1116 **4.18.3.4 type**

1117 The type of values for this attribute.

1118 **4.18.4 Outputs**

1119 **4.18.4.1 Iterator**

1120 See [iterator](#) in the [Inputs](#) section above.

1121 **4.18.4.2 value**

1122 Points to the variable where a pointer to the attribute value is to be returned. Note that the
1123 returned pointer points to the attribute's value in the list (no copy of the value is made) so
1124 that the caller does not need to do any special cleanup of the returned value's memory (it is
1125 cleaned up when the containing attribute list is deallocated).
1126 If this call returns an error, the output value is not changed.

1127 **4.18.5 Returns**

1128 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1129 returned.

1130 **4.18.6 Example**

```
1131 papi_status_t status;  
1132 papi_attribute_t **attrs = NULL;  
1133 papi_attribute_t **value = NULL;  
1134 ...  
1135 status = papiAttributeListGetCollection(attrs, NULL,  
1136                                     "media-col", &value);  
1137 ...  
1138 papiAttributeListFree(attrs);
```

1139 **4.18.7 See Also**

1140 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1141 **4.19 papiAttributeListGetMetadata**

1142 **4.19.1 Description**

1143 Get a meta-valued attribute's value from an attribute list.

1144 **4.19.2 Syntax**

```
1145 papi_status_t papiAttributeListGetMetadata(papi_attribute_t **attrs, void **iterator,  
1146 char *name, papi_metadata_t *value);
```

1147 **4.19.3 Inputs**

1148 **4.19.3.1 attrs**

1149 Points to an attribute list.

1150 **4.19.3.2 iterator**

1151 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1152 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1153 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1154 be passed in unchanged on subsequent calls to this function to get the remaining values.

1155 **4.19.3.3 name**

1156 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1157 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1158 found in the attribute list, but is not a PAPI_STRING, PAPI_NOT_POSSIBLE will be
1159 returned.

1160 **4.19.3.4 type**

1161 The type of values for this attribute.

1162 **4.19.4 Outputs**

1163 **4.19.4.1 Iterator**

1164 See [iterator](#) in the [Inputs](#) section above.

1165 **4.19.4.2 value**

1166 Points to the variable where the attribute value is to be returned. If this call returns an error,

1167 the output value is not changed.

1168 **4.19.5 Returns**

1169 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1170 returned.

1171 **4.19.6 Example**

```
1172 papi_status_t status;  
1173 papi_attribute_t **attrs = NULL;  
1174 papi_metadata_t value = PAPI_NO_VALUE;  
1175 ...  
1176 status = papiAttributeListGetMetadadata(attrs, NULL,  
1177                                         "media", &value);  
1178 ...  
1179 papiAttributeListFree(attrs);
```

1180 **4.19.7 See Also**

1181 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1182 **4.20 papiAttributeListFree**

1183 **4.20.1 Description**

1184 Frees an attribute list

1185 **4.20.2 Syntax**

```
1186 void papiAttributeListFree(papi_attribute_t **attrs);
```

1187 **4.20.3 Inputs**

1188 **4.20.3.1 attrs**

1189 Attribute list to be deallocated.

1190 **4.20.4 Outputs**

1191 none

1192 **4.20.5 Returns**

1193 none

1194 **4.20.6 Example**

```
1195 papi_attribute_t **attrs = NULL;  
1196 ...  
1197 papiAttributeListFree(attrs);
```

1198 **4.20.7 See Also**

1199 [papiAttributeListAddValue](#), [papiAttributeListAddString](#), [papiAttributeListAddInteger](#),
1200 [papiAttributeListAddBoolean](#), [papiAttributeListAddRange](#)
1201 [papiAttributeListAddResolution](#), [papiAttributeListAddDatetime](#),
1202 [papiAttributeListAddCollection](#), [papiAttributeListFromString](#), [papiAttributeListFree](#)

1203 **4.21 *papiAttributeListFind***

1204 **4.21.1 Description**

1205 Find an attribute in an attribute list.

1206 **4.21.2 Syntax**

```
1207 papi_attribute_t *papiAttributeListFind(papi_attribute_t **attrs, char *name);
```

1208 **4.21.3 Inputs**

1209 **4.21.3.1 *attrs***

1210 Points to an attribute list.

1211 **4.21.3.2 *name***

1212 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1213 supplied attribute list, PAPI_NOT_FOUND will be returned.

1214 **4.21.4 Outputs**

1215 none

1216 **4.21.5 Returns**

1217 Pointer to the named attribute found in the attribute list. The result will be deallocated
1218 when the containing attribute list is destroyed. NULL indicates that the specified attribute
1219 was not found

1220 **4.21.6 Example**

```
1221 papi_attribute_t **attrs = NULL;
```

Chapter 4: Attributes API

```
1222 papi_attribute_t *value;  
1223 ...  
1224 value = papiAttributeListFind(attrs, "job-name");  
1225 ...  
1226 papiAttributeListFree(attrs);
```

1227 **4.21.7 See Also**

1228 [papiAttributeListGetValue](#)

1229 **4.22 *papiAttributeListGetNext***

1230 **4.22.1 Description**

1231 Get the next attribute in an attribute list.

1232 **4.22.2 Syntax**

```
1233 papi_attribute_t **papiAttributeListGetNext(papi_attribute_t **attrs, void **iterator);
```

1234 **4.22.3 Inputs**

1235 **4.22.3.1 *attrs***

1236 Points to an attribute list.

1237 **4.22.3.2 *iterator***

1238 Pointer to an opaque (void*) iterator. This should be NULL to find the first attribute and
1239 then passed in unchanged on subsequent calls to this function.

1240 **4.22.4 Outputs**

1241 **4.22.4.1 *iterator***

1242 See [iterator](#) in the [Inputs](#) section above.

1243 **4.22.5 Returns**

1244 Pointer to the next attribute in the attribute list. The result will be deallocated when the
1245 containing attribute list is destroyed. NULL indicates that the end of the attribute list was
1246 reached

1247 **4.22.6 Example**

```
1248 papi_attribute_t **attrs = NULL;  
1249 papi_attribute_t *value;
```

```
1250 void *iterator = NULL;
1251 ...
1252 while ((value = papiAttributeGetNext(attrs, &iterator)) != NULL) {
1253     ...
1254 }
1255 ...
1256 papiAttributeListFree(attrs);
```

1257 **4.22.7 See Also**

1258 [papiAttributeListFind](#)

1259 **4.23 papiAttributeListFromString**

1260 **4.23.1 Description**

1261 Convert a string of text options to an attribute list. PAPI provides two functions which map
1262 job attributes to and from text options that are typically provided on the command-line by
1263 the user. This text encoding is also backwards-compatible with existing printing systems
1264 and is relatively simple to parse and generate. See [Attribute List Text Representation](#) for a
1265 definition of the string syntax.

1266 **4.23.2 Syntax**

```
1267 papi_status_t papiAttributeListFromString(papi_attribute_t*** attrs,
1268                                           int add_flags, char* buffer );
```

1269 **4.23.3 Inputs**

1270 **4.23.3.1 attrs**

1271 Points to an attribute list. If *attrs is NULL then this function will allocate the attribute list.

1272 **4.23.3.2 add_flags**

1273 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that
1274 indicates how to handle the request.

1275 **4.23.3.3 buffer**

1276 Points to text options.

1277 **4.23.4 Outputs**

1278 **4.23.4.1 attrs**

1279 The attribute list is updated.

1280 **4.23.5 Returns**

1281 If the text string is successfully converted to an attribute list, a value of PAPI_OK is
1282 returned. Otherwise an appropriate failure value is returned.

1283 **4.23.6 Example**

```
1284 papi_status_t status;  
1285 papi_attribute_t **attrs = NULL;  
1286 char *string = "copies=1 job-name=John\'s\ Really\040Nice\ Job";  
1287 ...  
1288 status = papiAttributeListFromString(attrs, PAPI_ATTR_EXCL, string);  
1289 ...  
1290 papiAttributeListFree(attrs);
```

1291 **4.23.7 See Also**

1292 [papiAttributeListFind](#)

1293 **4.24 papiAttributeListToString**

1294 **4.24.1 Description**

1295 Convert an attribute list to its text representation. The destination string is limited to at most
1296 (buflen - 1) bytes plus the trailing null byte.

1297 PAPI provides two functions which map job attributes to and from text options that are
1298 typically provided on the command-line by the user. This text encoding is also backwards-
1299 compatible with existing printing systems and is relatively simple to parse and generate.

1300 See [Attribute List Text Representation](#) for a definition of the string syntax.

1301 **4.24.2 Syntax**

```
1302 papi_status_t papiAttributeListToString(papi_attribute_t** attrs,  
1303                                         char* attr_delim, char* buffer,  
1304                                         size_t buflen );
```

1305 **4.24.3 Inputs**

1306 **4.24.3.1 attr**

1307 Points to an attribute list.

1308 **4.24.3.2 attr_delim**

1309 (optional) If not NULL, points to a string to be placed between attributes in the output
1310 buffer. If NULL, a space is used as the attribute delimiter.

1311 **4.24.3.3 buffer**

1312 Points to a string buffer to receive the to receive the text representation of the attribute list.

1313 **4.24.3.4 buflen**

1314 Specifies the length of the string buffer in bytes.

1315 **4.24.4 Outputs**

1316 **4.24.4.1 buffer**

1317 The buffer is filled with the text representation of the attribute list. The buffer will always
1318 be set to something by this function (buffer[0] = NULL in cases of an error).

1319 **4.24.5 Returns**

1320 If the attribute list is successfully converted to a text string, a value of PAPI_OK is
1321 returned. Otherwise an appropriate failure value is returned.

1322 **4.24.6 Example**

```
1323 papi_attribute_t **attrs = NULL;  
1324 char buffer[8192];  
1325 ...  
1326 papiAttributeListToString(attrs, NULL, buffer, sizeof (buffer));  
1327 ...  
1328 papiAttributeListFree(attrs);
```

1329 **4.24.7 See Also**

1330 [PapiAttributeListFromString](#)

1331 **Chapter 5: Service API**

1332 The service segment of the PAPI provides a means of creating, modifying, or destroying a
1333 context (or object) used to interact with a print service. This context is opaque to
1334 applications using it and may be used by implementations to store internal data such as file
1335 or socket descriptors, operation results, credentials, etc.

1336 **5.1 *papiServiceCreate***

1337 **5.1.1 Description**

1338 Create a print service handle to be used in subsequent calls. Memory is allocated and
1339 copies of the input arguments are created so that the handle can be used outside the scope of
1340 the input variables.
1341 The caller must call [papiServiceDestroy](#) when done in order to free the resources associated
1342 with the print service handle. This must be done even if the `papiServiceCreate` call failed,
1343 because a service creation failure may have resulted in a partial service context with
1344 additional error information.

1345 **5.1.2 Syntax**

```
1346 papi_status_t papiServiceCreate( papi_service_t *handle, char *service_name,  
1347                                 char *user_name, char *password,  
1348                                 int (*authCB)(papi_service_t svc),  
1349                                 papi_encryption_t encryption, void *app_data );
```

1350 **5.1.3 Inputs**

1351 **5.1.3.1 *service_name***

1352 (optional) Points to the name or URI of the service to use. A NULL value indicates that a
1353 “default service” should be used (the configuration of a default service is implementation-
1354 specific and may consist of environment variables, config files, etc. Default service
1355 selection is not addressed by this standard).

1356 **5.1.3.2 *user_name***

1357 (optional) Points to the name of the user who is making the requests. A NULL value
1358 indicates that the user name associated with the process in which the API call is made
1359 should be used.

1360 **5.1.3.3 *Password***

1361 (optional) Points to the password to be used to authenticate the user to the print service.

1362 **5.1.3.4 AuthCB**

1363 (optional) Points to a callback function to be used in authenticating the user to the print
1364 service if no password was supplied (or user input is required). A NULL value indicates
1365 that no callback should be made. The callback function should return 0 if the request is to
1366 be canceled and non-zero if new authentication information has been set.

1367 **5.1.3.5 Encryption**

1368 Specifies the encryption type to be used by the PAPI functions.

1369 **5.1.3.6 app_data**

1370 (optional) Points to application-specific data for use by the callback. The caller is
1371 responsible for allocating and freeing memory associated with this data.

1372 **5.1.4 Outputs**

1373 **5.1.4.1 handle**

1374 A print service handle to be used on subsequent API calls. The handle will always be set to
1375 something even if the function fails. In the event that the function fails, the handle may be
1376 set to NULL or it may be set to a valid handle that contains error information.

1377 **5.1.5 Returns**

1378 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1379 returned.

1380 **5.1.6 Example**

```
1381 papi_status_t status;  
1382 papi_service_t handle = NULL;  
1383 ...  
1384 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1385                             "user", "password", NULL,  
1386                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1387 ...  
1388 papiServiceDestroy(handle);
```

1389 **5.1.7 See Also**

1390 [papiServiceDestroy](#), [papiServiceGetStatusMessage](#), [papiServiceSetUserName](#),
1391 [papiServiceSetPassword](#), [papiServiceSetEncryption](#), [papiServiceSetAuthCB](#),
1392 [papiServiceSetAppData](#), [papiServiceGetStatusMessage](#)

1393 **5.2 *papiServiceDestroy***

1394 **5.2.1 Description**

1395 Destroy a print service handle and free the resources associated with it. This must be called
1396 even if the [papiServiceCreate](#) call failed, because there may be error information associated
1397 with the returned handle. If there is application data associated with the service handle, it is
1398 the caller's responsibility to free this memory.

1399 **5.2.2 Syntax**

```
1400 void papiServiceDestroy(papi_service_t handle);
```

1401 **5.2.3 Inputs**

1402 **5.2.3.1 *handle***

1403 The print service handle to be destroyed.

1404 **5.2.4 Outputs**

1405 None

1406 **5.2.5 Returns**

1407 None

1408 **5.2.6 Example**

```
1409 papi_status_t status;  
1410 papi_service_t handle = NULL;  
1411 ...  
1412 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1413                             "user", "password", NULL,  
1414                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1415 ...  
1416 papiServiceDestroy(handle);
```

1417 **5.2.7 See Also**

1418 [papiServiceCreate](#)

1419 **5.3 *papiServiceSetUserName***

1420 **5.3.1 Description**

1421 Set the user name in the print service handle to be used in subsequent calls. Memory is

1422 allocated and a copy of the input argument is created so that the handle can be used outside
1423 the scope of the input variable.

1424 **5.3.2 Syntax**

```
1425 papi_status_t papiServiceSetUserName( papi_service_t handle,  
1426                                     char* user_name );
```

1427 **5.3.3 Inputs**

1428 **5.3.3.1 handle**

1429 Handle to the print service to update.

1430 **5.3.3.2 user_name**

1431 Points to the name of the user who is making the requests. A NULL value indicates that
1432 the user name associated with the process in which the API call is made should be used.

1433 **5.3.4 Outputs**

1434 **5.3.4.1 handle**

1435 Handle remains unchanged, but it's contents may be updated.

1436 **5.3.5 Returns**

1437 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1438 returned.

1439 **5.3.6 Example**

```
1440 papi_status_t status;  
1441 papi_service_t handle = NULL;  
1442 ...  
1443 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1444                             "user", "password", NULL,  
1445                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1446 ...  
1447 status = papiServiceSetUserName(handle, "root");  
1448 ...  
1449 papiServiceDestroy(handle);
```

1450 **5.3.7 See Also**

1451 [papiServiceCreate](#), [papiServiceGetUserName](#), [papiServiceGetStatusMessage](#)

1452 **5.4 *papiServiceSetPassword***

1453 **5.4.1 Description**

1454 Set the password in the print service handle to be used in subsequent calls. Memory is
1455 allocated and a copy of the input argument is created so that the handle can be used outside
1456 the scope of the input variable.

1457 **5.4.2 Syntax**

```
1458 papi_status_t papiServiceSetPassword( papi_service_t handle, char* password);
```

1459 **5.4.3 Inputs**

1460 **5.4.3.1 *handle***

1461 Handle to the print service to update.

1462 **5.4.3.2 *password***

1463 Points to the password to be used to authenticate the user to the print service.

1464 **5.4.4 Outputs**

1465 **5.4.4.1 *handle***

1466 Handle remains unchanged, but it's contents may be updated.

1467 **5.4.5 Returns**

1468 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1469 returned.

1470 **5.4.6 Example**

```
1471 papi_status_t status;  
1472 papi_service_t handle = NULL;  
1473 ...  
1474 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1475                             "user", "password", NULL,  
1476                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1477 ...  
1478 status = papiServiceSetPassword(handle, "passsword");  
1479 ...  
1480 papiServiceDestroy(handle);
```

1481 **5.4.7 See Also**

1482 [papiServiceCreate](#), [papiServiceGetPassword](#), [papiServiceGetStatusMessage](#)

1483 **5.5 papiServiceSetEncryption**

1484 **5.5.1 Description**

1485 Set the encryption in the print service handle to be used in subsequent calls.

1486 **5.5.2 Syntax**

```
1487 papi_status_t papiServiceSetEncryption( papi_service_t handle,  
1488                                         papi_encryption_t encryption);
```

1489 **5.5.3 Inputs**

1490 **5.5.3.1 handle**

1491 Handle to the print service to update.

1492 **5.5.3.2 encryption**

1493 Specifies the encryption type to be used by the PAPI functions.

1494 **5.5.4 Outputs**

1495 **5.5.4.1 handle**

1496 Handle remains unchanged, but it's contents may be updated.

1497 **5.5.5 Returns**

1498 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1499 returned.

1500 **5.5.6 Example**

```
1501 papi_status_t status;  
1502 papi_service_t handle = NULL;  
1503 ...  
1504 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1505                               "user", "password", NULL,  
1506                               PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1507 ...  
1508 status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);  
1509 ...  
1510 papiServiceDestroy(handle);
```

1511 **5.5.7 See Also**

1512 [papiServiceCreate](#), [papiServiceGetEncryption](#), [papiServiceGetStatusMessage](#)

1513 **5.6 papiServiceSetAuthCB**

1514 **5.6.1 Description**

1515 Set the authorization callback function in the print service handle to be used in subsequent
1516 calls.

1517 **5.6.2 Syntax**

```
1518 papi_status_t papiServiceSetAuthCB( papi_service_t handle,  
1519                                     int (*authCB)(papi_service_t svc));
```

1520 **5.6.3 Inputs**

1521 **5.6.3.1 handle**

1522 Handle to the print service to update.

1523 **5.6.3.2 authCB**

1524 Points to a callback function to be used in authenticating the user to the print service if no
1525 password was supplied (or user input is required). A NULL value indicates that no callback
1526 should be made. The callback function should return 0 if the request is to be canceled and
1527 non-zero if new authentication information has been set.

1528 **5.6.4 Outputs**

1529 **5.6.4.1 handle**

1530 Handle remains unchanged, but it's contents may be updated.

1531 **5.6.5 Returns**

1532 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1533 returned.

1534 **5.6.6 Example**

```
1535 papi_status_t status;  
1536 papi_service_t handle = NULL;  
1537 ...  
1538 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1539                             "user", "password", NULL,  
1540                             PAPI_ENCRYPT_IF_REQUESTED, NULL);
```



```
1541 ...
1542 status = papiServiceSetAuthCB(handle, get_password_callback);
1543 ...
1544 papiServiceDestroy(handle);
```

1545 **5.6.7 See Also**

1546 [papiServiceCreate](#), [papiServiceGetStatusMessage](#)

1547 **5.7 papiServiceSetAppData**

1548 **5.7.1 Description**

1549 Set a pointer to some application-specific data in the print service. This data may be used
1550 by the authentication callback function. The caller is responsible for allocating and freeing
1551 memory associated with this data.

1552 **5.7.2 Syntax**

```
1553 papi_status_t papiServiceSetAppData( papi_service_t handle, void *app_data);
```

1554 **5.7.3 Inputs**

1555 **5.7.3.1 handle**

1556 Handle to the print service to update.

1557 **5.7.3.2 app_data**

1558 Points to application-specific data for use by the callback. The caller is responsible for
1559 allocating and freeing memory associated with this data.

1560 **5.7.4 Outputs**

1561 **5.7.4.1 handle**

1562 Handle remains unchanged, but it's contents may be updated.

1563 **5.7.5 Returns**

1564 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1565 returned.

1566 **5.7.6 Example**

```
1567 papi_status_t status;
1568 papi_service_t handle = NULL;
```

Chapter 5: Service API

```
1569 ...
1570 status = papiServiceCreate(&handle, "ipp://printserver:631",
1571                             "user", "password", NULL,
1572                             PAPI_ENCRYPT_IF_REQUESTED, NULL);
1573 ...
1574 status = papiServiceSetAppData(handle, app_data);
1575 ...
1576 papiServiceDestroy(handle);
```

1577 **5.7.7 See Also**

1578 [papiServiceCreate](#), [papiServiceGetAppData](#), [papiServiceGetStatusMessage](#)

1579 **5.8 papiServiceGetServiceName**

1580 **5.8.1 Description**

1581 Get the service name associated with the print service handle.

1582 **5.8.2 Syntax**

```
1583 char *papiServiceGetServiceName(papi_service_t handle);
```

1584 **5.8.3 Inputs**

1585 **5.8.3.1 handle**

1586 Handle to the print service.

1587 **5.8.4 Outputs**

1588 None

1589 **5.8.5 Returns**

1590 A pointer to the service name associated with the print service handle. The value returned
1591 will be deallocated upon destruction of the service handle.

1592 **5.8.6 Example**

```
1593 papi_status_t status;
1594 papi_service_t handle = NULL;
1595 char *service_name = NULL;
1596 ...
1597 service_name = papiServiceGetServiceName(handle);
1598 ...
1599 papiServiceDestroy(handle);
```

1600 **5.8.7 See Also**

1601 [papiServiceCreate](#)

1602 **5.9 papiServiceGetUserName**

1603 **5.9.1 Description**

1604 Get the user name associated with the print service handle.

1605 **5.9.2 Syntax**

```
1606 char *papiServiceGetUserName(papi_service_t handle);
```

1607 **5.9.3 Inputs**

1608 **5.9.3.1 handle**

1609 Handle to the print service.

1610 **5.9.4 Outputs**

1611 None

1612 **5.9.5 Returns**

1613 A pointer to the user name associated with the print service handle.

1614 **5.9.6 Example**

```
1615 papi_status_t status;  
1616 papi_service_t handle = NULL;  
1617 char *service_name = NULL;  
1618 ...  
1619 user_name = papiServiceGetUserName(handle);  
1620 ...  
1621 papiServiceDestroy(handle);
```

1622 **5.9.7 See Also**

1623 [papiServiceCreate](#), [papiServiceSetUserName](#)

1624 **5.10 papiServiceGetPassword**

1625 **5.10.1 Description**

1626 Get the password associated with the print service handle.

1627 **5.10.2 Syntax**

1628 `char *papiServiceGetPassword(papi_service_t handle);`

1629 **5.10.3 Inputs**

1630 **5.10.3.1 handle**

1631 Handle to the print service.

1632 **5.10.4 Outputs**

1633 None

1634 **5.10.5 Returns**

1635 A pointer to the password associated with the print service handle.

1636 **5.10.6 Example**

```
1637 papi_status_t status;  
1638 papi_service_t handle = NULL;  
1639 char *password = NULL;  
1640 ...  
1641 password = papiServiceGetPassword(handle);  
1642 ...  
1643 papiServiceDestroy(handle);
```

1644 **5.10.7 See Also**

1645 [papiServiceCreate](#), [papiServiceSetPassword](#)

1646 **5.11 papiServiceGetEncryption**

1647 **5.11.1 Description**

1648 Get the encryption associated with the print service handle.

1649 **5.11.2 Syntax**

1650 `papi_encryption_t papiServiceGetEncryption(papi_service_t handle);`

1651 **5.11.3 Inputs**

1652 **5.11.3.1 handle**

1653 Handle to the print service.

1654 **5.11.4 Outputs**

1655 None

1656 **5.11.5 Returns**

1657 The type of encryption associated with the print service handle.

1658 **5.11.6 Example**

```
1659 papi_status_t status;  
1660 papi_service_t handle = NULL;  
1661 papi_encryption_t encryption;  
1662 ...  
1663 encryption = papiServiceGetEncryption(handle);  
1664 ...  
1665 papiServiceDestroy(handle);
```

1666 **5.11.7 See Also**

1667 [papiServiceCreate](#), [papiServiceSetEncryption](#)

1668 **5.12 *papiServiceGetAppData***

1669 **5.12.1 Description**

1670 Get a pointer to the application-specific data associated with the print service handle.

1671 **5.12.2 Syntax**

```
1672 void *papiServiceGetAppData(papi_service_t handle);
```

1673 **5.12.3 Inputs**

1674 **5.12.3.1 *handle***

1675 Handle to the print service.

1676 **5.12.4 Outputs**

1677 None

1678 **5.12.5 Returns**

1679 A pointer to the application-specific data associated with the print service handle.

1680 **5.12.6 Example**

```
1681 papi_status_t status;  
1682 papi_service_t handle = NULL;  
1683 void app_data = NULL;  
1684 ...  
1685 app_data = papiServiceGetAppData(handle);  
1686 ...  
1687 papiServiceDestroy(handle);
```

1688 **5.12.7 See Also**

1689 [papiServiceCreate](#), [papiServiceSetAppData](#)

1690 **5.13 papiServiceGetAttributeList**

1691 **5.13.1 Description**

1692 Retrieve an attribute list from the print service. This attribute list contains service specific
1693 attributes describing service and implementation specific features.

1694 **5.13.2 Syntax**

```
1695 papi_attribute_t **papiServiceGetAttributeList(papi_service_t handle);
```

1696 **5.13.3 Inputs**

1697 **5.13.3.1 handle**

1698 Handle to the print service.

1699 **5.13.4 Outputs**

1700 None

1701 **5.13.5 Returns**

1702 An attribute list associated with the print service handle. The attribute list is destroyed
1703 when the service handle is destroyed.

1704 **5.13.6 Example**

```
1705 papi_status_t status;  
1706 papi_service_t handle = NULL;  
1707 papi_attribute_t **attributes = NULL;  
1708 ...  
1709 attributes = papiServiceGetAttributeList(handle);  
1710 ...
```

```
1711 papiServiceDestroy(handle);
```

1712 **5.13.7 See Also**

1713 [papiServiceCreate](#), [papiServiceDestroy](#)

1714 **5.14 papiServiceGetStatusMessage**

1715 **5.14.1 Description**

1716 Get the message associated with the status of the last operation performed. The status
1717 message returned from this function may be more detailed than the status message returned
1718 from `papiStatusString` (if the print service supports returning more detailed error messages).
1719 The returned message will be localized in the language of the submitter of the original
1720 operation.

1721 **5.14.2 Syntax**

```
1722 Char *papiServiceGetStatusMessage(papi_service_t handle);
```

1723 **5.14.3 Inputs**

1724 **5.14.3.1 handle**

1725 Handle to the print service.

1726 **5.14.4 Outputs**

1727 None

1728 **5.14.5 Returns**

1729 Pointer to the message associated with the print service handle.

1730 **5.14.6 Example**

```
1731 papi_status_t status;  
1732 papi_service_t handle = NULL;  
1733 char *message = NULL;  
1734 ...  
1735 message = papiServiceGetStatusMessage(handle);  
1736 ...  
1737 papiServiceDestroy(handle);
```

1738 **5.14.7 See Also**

1739 [papiServiceCreate](#), [papiServiceSetUserName](#), [papiServiceSetPassword](#),

Chapter 5: Service API

1740 [papiServiceSetEncryption](#), [papiServiceSetAuthCB](#), [papiServiceSetAppData](#), [Printer API](#),
1741 [Attributes API](#), [Job API](#)

1742 **Chapter 6: Printer API**

1743 The printer segment of the PAPI provides a means of interacting with printer objects
1744 contained in a print service. This interaction can include listing, querying, modifying,
1745 pausing, and releasing the printer objects themselves. It can also include clearing all jobs
1746 from a printer object or enumerating all jobs associated with a printer object.

1747 The [papiPrinterQuery](#) function queries all/some of the attributes of a printer object. It
1748 returns a list of printer attributes. A successful call to [papiPrinterQuery](#) is typically followed
1749 by code which examines and processes the returned attributes. When the calling program is
1750 finished with the printer object and its attributes, it should then call [papiPrinterFree](#) to
1751 delete the returned results.

1752 Printers can be found via calls to [papiPrintersList](#). A successful call to [papiPrintersList](#) is
1753 typically followed by code to iterate through the list of returned printers, possibly querying
1754 each ([papiPrinterQuery](#)) for further information (e.g. to restrict what printers get displayed
1755 for a particular user/request). When the calling program is finished with the list of printer
1756 objects, it should then call [papiPrinterListFree](#) to free the returned results.

1757 **6.1 papiPrintersList**

1758 **6.1.1 Description**

1759 List all printers known by the print service which match the specified filter.
1760 Depending on the functionality of the target service's "printer directory", the returned list
1761 may be limited to only printers managed by a particular server or it may include printers
1762 managed by other servers.

1763 **6.1.2 Syntax**

```
1764 papi_status_t papiPrintersList(papi_service_t handle, char *requested_attrs[],  
1765                               papi_filter_t *filter, papi_printer_t **printers );
```

1766 **6.1.3 Inputs**

1767 **6.1.3.1 handle**

1768 Handle to the print service.

1769 **6.1.3.2 requested_attrs**

1770 (optional) NULL terminated array of attributes to be queried. If NULL is passed then all
1771 attributes are queried. (NOTE: The printer may return more attributes than you requested.
1772 This is merely an advisory request that may reduce the amount of data returned if the
1773 printer/server supports it.)

Chapter 6: Printer API

1774 **6.1.3.3 filter**

1775 (optional) Pointer to a filter to limit the number of printers returned on the list request. See
1776 for details. If NULL is passed then all known printers are listed.

1777 **6.1.4 Outputs**

1778 **6.1.4.1 printers**

1779 List of printer objects that matched the filter criteria. The resulting list of printer objects
1780 must be deallocated by the caller using [papiPrinterListFree\(\)](#).

1781 **6.1.5 Returns**

1782 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1783 returned.

1784 **6.1.6 Example**

```
1785 papi_status_t status;  
1786 papi_service_t handle = NULL;  
1787 char *req_attrs[] = { "printer-name", "printer-uri", NULL };  
1788 papi_filter_t filter;  
1789 papi_printer_t *printers = NULL;  
1790 ...  
1791 /* Select local printers (non-remote) that support color */  
1792 filter.type = PAPI_FILTER_BITMASK;  
1793 filter.filter.bitmask.mask = PAPI_PRINTER_REMOTE |  
1794 PAPI_PRINTER_COLOR;  
1795 filter.filter.bitmask.value = PAPI_PRINTER_COLOR;  
1796 ...  
1797 status = papiPrinterList(handle, req_attrs, filter, &printers);  
1798 ...  
1799 if (printers != NULL) {  
1800     int i;  
1801     for (i = 0; printers[i] != NULL; i++) {  
1802         ...  
1803     }  
1804     papiPrinterListFree(printers);  
1805 }  
1806 ...  
1807 ...  
1808 papiServiceDestroy(handle);
```

1809 **6.1.7 See Also**

1810 [papiPrinterListFree](#), [papiPrinterQuery](#)

1811 **6.2 papiPrinterQuery**

1812 **6.2.1 Description**

1813 Queries some or all the attributes of the specified printer object. This includes attributes
1814 representing information and capabilities of the printer. The caller may use this information
1815 to determine which print options to present to the user. How the attributes are obtained (e.g.
1816 from a static database, from a dialog with the hardware, from a dialog with a driver, etc.) is
1817 implementation specific and is beyond the scope of this standard. The call optionally
1818 includes "context" information which specifies job attributes that provide a context that can
1819 be used by the print service to construct capabilities information.

1820 **6.2.2 Semantics Reference**

1821 Get-Printer-Attributes in [RFC2911], section 3.2.5

1822 **6.2.3 Syntax**

```
1823 papi_status_t papiPrinterQuery(papi_service_t handle, char *name,  
1824                               char *requested_attrs[], papi_attribute_t **job_attrs,  
1825                               papi_printer_t *printer );
```

1826 **6.2.4 Inputs**

1827 **6.2.4.1 handle**

1828 Handle to the print service to use.

1829 **6.2.4.2 name**

1830 The name or URI of the printer to query.

1831 **6.2.4.3 requested_attrs**

1832 (optional) NULL terminated array of attributes to be queried. If NULL is passed then all
1833 attributes are queried. (NOTE: The printer may return more attributes than you requested.
1834 This is merely an advisory request that may reduce the amount of data returned if the
1835 printer/server supports it.)

1836 **6.2.4.4 job_attrs**

1837 (optional) NULL terminated array of job attributes in the context of which the capabilities
1838 information is to be constructed. In other words, the returned printer attributes represent the
1839 capabilities of the printer given that these specified job attributes are requested. This allows
1840 for more accurate information to be retrieved by the caller for a specific job (e.g. "if the job
1841 is printed on A4 size media then duplex output is not available"). If NULL is passed then
1842 the full capabilities of the printer are queried.

1843 Support for this argument is optional. If the underlying print system does not have access to
1844 capabilities information bound by job context, then this argument may be ignored. But if
1845 the calling application will be using the returned information to build print job data, then it
1846 is always advisable to specify the job context attributes. The more context information
1847 provided, the more accurate capabilities information is likely to be returned from the print
1848 system.

1849 **6.2.5 Outputs**

1850 **6.2.5.1 printer**

1851 Pointer to a printer object containing the requested attributes.

1852 **6.2.6 Returns**

1853 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1854 returned.

1855 **6.2.7 Example**

```
1856 papi_status_t status;  
1857 papi_service_t handle = NULL;  
1858 char *req_attrs[] = { "printer-name", "printer-uri",  
1859     "printer-state", "printer-state-reasons", NULL };  
1860 papi_attribute_t **job_attrs = NULL;  
1861 papi_printer_t printer = NULL;  
1862 ...  
1863 papiAttributeListAddString(&job_attrs, PAPI_EXCL,  
1864     "media", "legal");  
1865 ...  
1866 status = papiPrinterQuery(handle, "ipp://server/printers/queue",  
1867     req_attrs, job_attrs, &printer);  
1868 papiAttributeListFree(job_attrs);  
1869 ...  
1870 if (printer != NULL) {  
1871     /* process the printer object */  
1872     ...  
1873     papiPrinterFree(printer);  
1874 }  
1875 ...  
1876 papiServiceDestroy(handle);
```

1877 **6.2.8 See Also**

1878 [papiPrintersList](#), [papiPrinterFree](#)

1879 **6.3 *papiPrinterModify***

1880 **6.3.1 Description**

1881 Modifies some or all the attributes of the specified printer object. Upon successful
1882 completion, the function will return a handle to an object representing the updated printer.

1883 **6.3.2 Semantics Reference**

1884 Set-Job-Attributes in [RFC3380], section 4.2

1885 **6.3.3 Syntax**

```
1886 papi_status_t papiPrinterModify(papi_service_t handle, char *printer_name,  
1887 papi_attribute_t **attrs, papi_printer_t *printer );
```

1888 **6.3.4 Inputs**

1889 **6.3.4.1 *handle***

1890 Handle to the print service to use.

1891 **6.3.4.2 *name***

1892 The name or URI of the printer to be modified.

1893 **6.3.4.3 *attrs***

1894 Attributes to be modified. Any attributes not specified are left unchanged. Attributes can be
1895 deleted from the print service's printer object through the use of the PAPI_DELETE
1896 attribute metadata type.

1897 **6.3.5 Outputs**

1898 **6.3.5.1 *printer***

1899 The modified printer object.

1900 **6.3.6 Returns**

1901 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1902 returned.

1903 **6.3.7 Example**

```
1904 papi_status_t status;  
1905 papi_service_t handle = NULL;  
1906 papi_printer_t printer = NULL;
```

Chapter 6: Printer API

```
1907 papi_attribute_t **attrs = NULL;
1908 ...
1909 papiAttributeListAddString(&attrs, PAPI_EXCL,
1910     "printer-location", "Bldg 17/Room 234");
1911 papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
1912     "sample-data", PAPI_DELETE);
1913 ...
1914 status = papiPrinterModify(handle, "printer", attrs, &printer);
1915 ...
1916 if (printer != NULL) {
1917     /* process the printer */
1918     ...
1919     papiPrinterFree(printer);
1920 }
1921 ...
1922 papiServiceDestroy(handle);
```

1923 **6.3.8 See Also**

1924 [PapiPrinterQuery](#), [papiPrinterAdd](#), [papiPrinterRemove](#), [papiPrinterFree](#)

1925 **6.4 papiPrinterAdd**

1926 **6.4.1 Description**

1927 Creates a printer object with some or all the attributes specified. Upon successful
1928 completion, the function will return a handle to an object representing the created printer.

1929 **6.4.2 Semantics Reference**

1930 Set-Job-Attributes in [RFC3380], section 4.2

1931 **6.4.3 Syntax**

```
1932 papi_status_t papiPrinterAdd(papi_service_t handle, char *printer_name,
1933     papi_attribute_t **attrs, papi_printer_t *printer );
```

1934 **6.4.4 Inputs**

1935 **6.4.4.1 handle**

1936 Handle to the print service on which to create the printer object.

1937 **6.4.4.2 name**

1938 The name or URI of the printer to be created

1939 **6.4.4.3 attrs**

1940 Attributes to associate with the printer object being created. The print service may not
1941 honor all requested attributes and it may also add attributes of it's own during printer
1942 creation.

1943 **6.4.5 Outputs**

1944 **6.4.5.1 printer**

1945 The created printer object.

1946 **6.4.6 Returns**

1947 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1948 returned.

1949 **6.4.7 Example**

```
1950 papi_status_t status;  
1951 papi_service_t handle = NULL;  
1952 papi_printer_t printer = NULL;  
1953 papi_attribute_t **attrs = NULL;  
1954 ...  
1955 papiAttributeListAddString(&attrs, PAPI_ATTR_EXCL,  
1956     "printer-location", "Bldg 17/Room 234");  
1957 papiAttributeListAddString(&attrs, PAPI_ATTR_APPEND,  
1958     "document-format-supported", "application/ps");  
1959 papiAttributeListAddString(&attrs, PAPI_ATTR_APPEND,  
1960     "document-format-supported", "text/plain");  
1961 papiAttributeListAddInteger(&attrs, PAPI_ATTR_APPEND,  
1962     "copies-default", 3);  
1963 ...  
1964 status = papiPrinterModify(handle, "ipp://server/printers/triplicate",  
1965     attrs, &printer);  
1966 papiAttributeListFree(attrs);  
1967 ...  
1968 if (status != PAPI_OK) {  
1969     /* report a failure */  
1970 }  
1971 ...  
1972 if (printer != NULL) {  
1973     /* process the printer */  
1974     ...  
1975     papiPrinterFree(printer);  
1976 }  
1977 ...  
1978 papiServiceDestroy(handle);
```

1979 **6.4.8 See Also**

1980 [papiPrinterQuery](#), [papiPrinterModify](#), [papiPrinterRemove](#), [papiPrinterFree](#)

1981 **6.5 *papiPrinterRemove***

1982 **6.5.1 Description**

1983 Removes a printer object from a print service.

1984 **6.5.2 Semantics Reference**

1985 Set-Job-Attributes in [RFC3380], section 4.2

1986 **6.5.3 Syntax**

1987 `papi_status_t papiPrinterRemove(papi_service_t handle, char *printer_name);`

1988 **6.5.4 Inputs**

1989 **6.5.4.1 *handle***

1990 Handle to the print service from which to remove the printer object.

1991 **6.5.4.2 *name***

1992 The name or URI of the printer to be removed

1993 **6.5.5 Outputs**

1994 None

1995 **6.5.6 Returns**

1996 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1997 returned.

1998 **6.5.7 Example**

```
1999 papi_status_t status;  
2000 papi_service_t handle = NULL;  
2001 ...  
2002 status = papiPrinterRemove(handle, "ipp://server/printers/goodbye");  
2003 if (status != PAPI_OK) {  
2004     /* report a failure */  
2005 }  
2006 ...  
2007 papiServiceDestroy(handle);
```


2008 **6.5.8 See Also**

2009 [papiPrinterQuery](#), [papiPrinterModify](#), [papiPrinterAdd](#), [papiPrinterFree](#)

2010 **6.6 papiPrinterPause**

2011 **6.6.1 Description**

2012 Stops the printer object from scheduling jobs to be printed. Depending on the
2013 implementation, this operation may also stop the printer from processing the current job(s).
2014 This operation is optional and may not be supported by all printers/servers. Use
2015 [papiPrinterResume](#) to undo the effects of this operation.

2016 **6.6.2 Semantics Reference**

2017 Pause-Printer in [RFC2911], section 3.2.7

2018 **6.6.3 Syntax**

2019 `papi_status_t papiPrinterPause(papi_service_t handle, char *name,`
2020 `char *message);`

2021 **6.6.4 Inputs**

2022 **6.6.4.1 handle**

2023 Handle to the print service to use.

2024 **6.6.4.2 name**

2025 The name or URI of the printer to operate on.

2026 **6.6.4.3 message**

2027 (optional) An explanatory message to be associated with the paused printer. This message
2028 may be ignored if the underlying print system does not support associating a message with
2029 a paused printer.

2030 **6.6.5 Outputs**

2031 None

2032 **6.6.6 Returns**

2033 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2034 returned.

2035 **6.6.7 Example**

```
2036 papi_status_t status;  
2037 papi_service_t handle = NULL;  
2038 ...  
2039 status = papiPrinterPause(handle, "printer", "because I can");  
2040 ...  
2041 papiServiceDestroy(handle);
```

2042 **6.6.8 See Also**

2043 [papiPrinterResume](#)

2044 **6.7 papiPrinterResume**

2045 **6.7.1 Description**

2046 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the effects of
2047 [papiPrinterPause](#)). This operation is optional and may not be supported by all
2048 printers/servers, but it must be supported if [papiPrinterPause](#) is supported.

2049 **6.7.2 Semantics Reference**

2050 Resume-Printer in [RFC2911], section 3.2.8

2051 **6.7.3 Syntax**

```
2052 papi_status_t papiPrinterResume(papi_service_t handle, char *name);
```

2053 **6.7.4 Inputs**

2054 **6.7.4.1 handle**

2055 Handle to the print service to use.

2056 **6.7.4.2 name**

2057 The name or URI of the printer to operate on.

2058 **6.7.5 Outputs**

2059 None

2060 **6.7.6 Returns**

2061 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2062 returned.

2063 **6.7.7 Example**

```
2064 papi_status_t status;  
2065 papi_service_t handle = NULL;  
2066 ...  
2067 status = papiPrinterResume(handle, "printer");  
2068 ...  
2069 papiServiceDestroy(handle);
```

2070 **6.7.8 See Also**

2071 [papiPrinterPause](#)

2072 **6.8 papiPrinterEnable**

2073 **6.8.1 Description**

2074 Requests that the printer enable job creation (queueing) (i.e. it undoes the effects of
2075 [papiPrinterDisable](#)). This operation is optional and may not be supported by all
2076 printers/servers, but it must be supported if [papiPrinterDisable](#) is supported.

2077 **6.8.2 Semantics Reference**

2078 Resume-Printer in [RFC2911], section 3.2.8

2079 **6.8.3 Syntax**

```
2080 papi_status_t papiPrinterEnable(papi_service_t handle, char *name);
```

2081 **6.8.4 Inputs**

2082 **6.8.4.1 handle**

2083 Handle to the print service to use.

2084 **6.8.4.2 name**

2085 The name or URI of the printer to operate on.

2086 **6.8.5 Outputs**

2087 None

2088 **6.8.6 Returns**

2089 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2090 returned.

2091 **6.8.7 Example**

```
2092 papi_status_t status;  
2093 papi_service_t handle = NULL;  
2094 ...  
2095 status = papiPrinterEnable(handle, "printer");  
2096 ...  
2097 papiServiceDestroy(handle);
```

2098 **6.8.8 See Also**

2099 [papiPrinterDisable](#)

2100 **6.9 papiPrinterDisable**

2101 **6.9.1 Description**

2102 Requests that the printer disable job creation (queueing) (i.e. it undoes the effects of
2103 [papiPrinterEnable](#)). This operation is optional and may not be supported by all
2104 printers/servers, but it must be supported if [papiPrinterEnable](#) is supported.

2105 **6.9.2 Semantics Reference**

2106 Resume-Printer in [RFC2911], section 3.2.8

2107 **6.9.3 Syntax**

```
2108 papi_status_t papiPrinterDisable(papi_service_t handle, char *name, char *message);
```

2109 **6.9.4 Inputs**

2110 **6.9.4.1 handle**

2111 Handle to the print service to use.

2112 **6.9.4.2 name**

2113 The name or URI of the printer to operate on.

2114 **6.9.4.3 Message**

2115 An optional reason for disabling the print queue.

2116 **6.9.5 Outputs**

2117 None

2118 **6.9.6 Returns**

2119 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2120 returned.

2121 **6.9.7 Example**

```
2122 papi_status_t status;  
2123 papi_service_t handle = NULL;  
2124 ...  
2125 status = papiPrinterEnable(handle, "printer", "because it's tuesday");  
2126 ...  
2127 papiServiceDestroy(handle);
```

2128 **6.9.8 See Also**

2129 [papiPrinterDisable](#)

2130 **6.10 papiPrinterPurgeJobs**

2131 **6.10.1 Description**

2132 Remove all jobs from the specified printer object regardless of their states. This includes
2133 removing jobs that have completed and are being retained(if any). This operation is optional
2134 and may not be supported by all printers/servers.

2135 **6.10.2 Semantics Reference**

2136 Purge-Jobs in [RFC2911], section 3.2.9

2137 **6.10.3 Syntax**

```
2138 papi_status_t papiPrinterPurgeJobs(papi_service_t handle, char *name,  
2139 papi_job_t **jobs);
```

2140 **6.10.4 Inputs**

2141 **6.10.4.1 handle**

2142 Handle to the print service to use.

2143 **6.10.4.2 name**

2144 The name or URI of the printer to operate on.

2145 **6.10.5 Outputs**2146 **6.10.5.1 jobs**

2147 (optional) Pointer to a list of purged jobs with the identifying information (job-id/job-uri),
 2148 success/fail, and possibly a detailed message. If NULL is passed then no job list is returned.
 2149 Support for the returned job list is optional and may not be supported by all
 2150 implementations (if not supported, the function completes with PAPI_OK_SUBST but no
 2151 list is returned).

2152 **6.10.6 Returns**

2153 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
 2154 returned.

2155 **6.10.7 Example**

```

2156 #include "papi.h"
2157
2158 papi_status_t status;
2159 papi_service_t handle = NULL;
2160 char *service_name = "ipp://printserv:631";
2161 char *user_name = "pappy";
2162 char *password = "goober";
2163 char *printer_name = "my-printer";
2164 papi_job_t *jobs = NULL;
2165
2166 status = papiServiceCreate(handle, service_name, user_name,
2167                             password, NULL, PAPI_ENCRYPT_IF_REQUESTED,
2168                             NULL);
2169 if (status != PAPI_OK) {
2170     /* handle the error */
2171     ...
2172 }
2173
2174 status = papiPrinterPurgeJobs(handle, printer_name, &jobs);
2175 if (status != PAPI_OK) {
2176     /* handle the error */
2177     fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
2178             papiServiceGetStatusMessage(handle));
2179     ...
2180 }
2181
2182 if (jobs != NULL) {
2183     int i;
2184
2185     for(i=0; jobs[i] != NULL; i++) {
2186         /* process the job */
2187         ...
2188     }

```

```
2189     papiJobListFree (jobs) ;
2190 }
2191 ...
2192
2193 papiServiceDestroy (handle) ;
```

2194 **6.10.8 See Also**

2195 [papiJobCancel](#), [papiJobListFree](#)

2196 **6.11 papiPrinterListJobs**

2197 **6.11.1 Description**

2198 List print job(s) associated with the specified printer.

2199 **6.11.2 Semantics Reference**

2200 Get-Jobs in [RFC2911], section 3.2.6

2201 **6.11.3 Syntax**

```
2202 papi_status_t papiPrinterListJobs(papi_service_t handle, char *printer,
2203                                   char *requested_attrs[], int type_mask,
2204                                   int max_num_jobs, papi_job_t **jobs );
```

2205 **6.11.4 Inputs**

2206 **6.11.4.1 handle**

2207 Handle to the print service to use.

2208 **6.11.4.2 name**

2209 The name or URI of the printer to query.

2210 **6.11.4.3 requested_attrs**

2211 (optional) NULL terminated array of attributes to be queried. If NULL is passed then all
2212 available attributes are queried. (NOTE: The printer may return more attributes than you
2213 requested. This is merely an advisory request that may reduce the amount of data returned
2214 if the printer/server supports it.)

2215 **6.11.4.4 type_mask**

2216 A bit mask which determines what jobs will get returned. The following constants can be
2217 bitwise-OR-ed together to select which types of jobs to list:

Chapter 6: Printer API

```
2218     #define PAPI_LIST_JOBS_OTHERS    0x0001 /* return jobs other than
2219                                     those submitted by the
2220                                     user name associated with
2221                                     the handle */
2222     #define PAPI_LIST_JOBS_COMPLETED  0x0002 /* return completed jobs */
2223     #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
2224                                     jobs */
2225     #define PAPI_LIST_JOBS_ALL        0xFFFF /* return all jobs */
```

2226 **6.11.4.5 max_num_jobs**

2227 Limit to the number of jobs returned. If 0 is passed, then there is no limit to the number of
2228 jobs which may be returned.

2229 **6.11.5 Outputs**

2230 **6.11.5.1 jobs**

2231 List of job objects returned.

2232 **6.11.6 Returns**

2233 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2234 returned.

2235 **6.11.7 Example**

```
2236 papi_status_t status;
2237 papi_service_t handle = NULL;
2238 papi_job_t *jobs = NULL;
2239 char *job_attrs[] = {
2240     "job-id", "job-name", "job-originating-user-name",
2241     "job-state", "job-state-reasons", "job-state-message" };
2242 ...
2243 status = papiPrinterListJobs(handle, printer_name, job_attrs,
2244                             PAPI_LIST_JOBS_ALL, 0, &jobs);
2245 ...
2246 if (jobs != NULL) {
2247     int i;
2248
2249     for(i = 0; jobs[i] != NULL; i++) {
2250         /* process the job */
2251         ...
2252     }
2253     papiJobListFree(jobs);
2254 }
2255 ...
2256 papiServiceDestroy(handle);
```


2257 **6.11.8 See Also**

2258 [papiJobQuery](#), [papiJobListFree](#)

2259 **6.12 *papiPrinterGetAttributeList***

2260 **6.12.1 Description**

2261 Get the attribute list associated with a printer object.

2262 This function retrieves an attribute list from a printer object returned in a previous call.

2263 Printer objects are returned as the result of operations performed by [papiPrintersList](#),

2264 [papiPrinterQuery](#), and [papiPrinterModify](#).

2265 **6.12.2 Syntax**

```
2266 papi_attribute_t **papiPrinterGetAttributeList(papi_printer_t printer);
```

2267 **6.12.3 Inputs**

2268 **6.12.3.1 *printer***

2269 Handle of the printer object.

2270 **6.12.4 Outputs**

2271 none

2272 **6.12.5 Returns**

2273 Pointer to the attribute list associated with the printer object. This attribute list is

2274 deallocated when the printer object it was retrieved from is deallocated using

2275 [papiPrinterFree](#)(printer).

2276 **6.12.6 Example**

```
2277 papi_attribute_t **attrs = NULL;
2278 papi_printer_t printer = NULL;
2279 ...
2280 attrs = papiPrinterGetAttributeList(printer);
2281 ...
2282 papiPrinterFree(printer);
```

2283 **6.12.7 See Also**

2284 [papiPrintersList](#), [papiPrinterQuery](#), [papiPrinterModify](#)

2285 **6.13 *papiPrinterFree***

2286 **6.13.1 Description**

2287 Free a printer object.

2288 **6.13.2 Syntax**

```
2289 void papiPrinterFree(papi_printer_t printer);
```

2290 **6.13.3 Inputs**

2291 **6.13.3.1 *printer***

2292 Handle of the printer object to free.

2293 **6.13.4 Outputs**

2294 none

2295 **6.13.5 Returns**

2296 none

2297 **6.13.6 Example**

```
2298 papi_printer_t printer = NULL;  
2299 ...  
2300 papiPrinterFree(printer);
```

2301 **6.13.7 See Also**

2302 [papiPrinterQuery](#), [papiPrinterModify](#)

2303 **6.14 *papiPrinterListFree***

2304 **6.14.1 Description**

2305 Free a list of printer objects.

2306 **6.14.2 Syntax**

```
2307 void papiPrinterListFree(papi_printer_t *printers);
```

2308 **6.14.3 Inputs**

2309 **6.14.3.1 printers**

2310 Pointer to the printer object list to free.

2311 **6.14.4 Outputs**

2312 none

2313 **6.14.5 Returns**

2314 none

2315 **6.14.6 Example**

```
2316 papi_printer_t* printers = NULL;  
2317 ...  
2318 papiPrinterListFree(printers);
```

2319 **6.14.7 See Also**

2320 [papiPrintersList](#)

2321 **Chapter 7: Job API**

2322 The job segment of the PAPI provides a means of interacting with job objects contained in
2323 a print service. This interaction can include listing, querying, creating, modifying,
2324 canceling, holding, releasing, and restarting the job objects themselves.

2325 The [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobStreamOpen](#), and
2326 [papiJobStreamClose](#) functions provide a means of creating job objects under a print service.
2327 The [papiJobValidate](#) function can be used to determine if a job submission will be
2328 successful. Each of these functions results in a job object with an attribute list that can be
2329 queried to determine what the resulting job looks like.

2330 The [papiJobQuery](#) function queries all/some of the attributes of a job. A successful call to
2331 [papiJobQuery](#) is typically followed by code which examines and processes the returned
2332 attributes. When the calling program is finished with the job object and it's attributes, it
2333 should then call [papiJobFree](#) to delete the returned results.

2334 Jobs and job state can be modified through the use of [papiJobModify](#), [papiJobHold](#),
2335 [papiJobRelease](#), and [papiJobRestart](#). The [papiJobModify](#) call returns a job object that
2336 contains a representation of the modified job. The job object's attribute list can be queried
2337 to determin what the resulting job looks like. When the calling program is finished with the
2338 job object and it's attributes, it should then call [papiJobFree](#) to delete the returned results.

2339 **7.1 *papiJobSubmit***

2340 **7.1.1 Description**

2341 Submits a print job having the specified attributes to the specified printer. This interface
2342 copies the specified print files before returning to the caller (contrast to
2343 [papiJobSubmitByReference](#)). The caller must call [papiJobFree](#) when done in order to free
2344 the resources associated with the returned job object. Attributes of the print job may be
2345 passed in the `job_attributes` argument and/or in a job ticket (using the `job_ticket` argument).
2346 If both are specified, the attributes in the `job_attributes` list will be applied to the `job_ticket`
2347 attributes and the resulting attribute set will be used.

2348 **7.1.2 Semantics Reference**

2349 Print-Job in [RFC2911], section 3.2.1

2350 **7.1.3 Syntax**

```
2351 papi_status_t papiJobSubmit(papi_service_t handle, char *printer_name,  
2352                             papi_attribute_t **job_attributes,  
2353                             papi_job_ticket_t *job_ticket,  
2354                             char **file_names, papi_job_t *job );
```

2355 **7.1.4 Inputs**

2356 **7.1.4.1 handle**

2357 Handle to the print service to use.

2358 **7.1.4.2 printer_name**

2359 Pointer to the name of the printer to which the job is to be submitted.

2360 **7.1.4.3 job_attributes**

2361 (optional) The list of attributes describing the job and how it is to be printed. If options are
2362 specified here and also in the job ticket data, the value specified here takes precedence. If
2363 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2364 job.

2365 **7.1.4.4 job_ticket**

2366 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2367 job ticket is used with the job. Whether the implementation passes both the attributes and
2368 the job ticket to the server/printer, or merges them to some print protocol or internal
2369 representation depends on the implementation.

2370 **7.1.4.5 file_names**

2371 NULL terminated list of pointers to names of files to print. If more than one file is
2372 specified, the files will be treated by the print system as separate "documents" for things
2373 like page breaks and separator sheets, but they will be scheduled and printed together as one
2374 job and the specified attributes will apply to all the files.
2375 These file names may contain absolute path names or relative path names (relative to the
2376 current path). The implementation MUST copy the file contents before returning.

2377 **7.1.5 Outputs**

2378 **7.1.5.1 job**

2379 The resulting job object representing the submitted job. The caller must deallocate this
2380 object using [papiJobFree\(\)](#) when finished using it.

2381 **7.1.6 Returns**

2382 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2383 returned.

2384 **7.1.7 Example**

2385 `papi_status_t status;`

Chapter 7: Job API

```
2386 papi_service_t handle = NULL;
2387 papi_attribute_t **attrs = NULL;
2388 papi_job_ticket_t *ticket = NULL;
2389 char *files[] = { "/etc/motd", NULL };
2390 papi_job_t job = NULL;
2391 ...
2392 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
2393                             PAPI_STRING, 1, "test job");
2394 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
2395                              PAPI_INTEGER, 4);
2396 ...
2397 status = papiJobSubmit(handle, "printer", attrs, ticket, files, &job);
2398 papiAttributeListFree(attrs);
2399 ...
2400 if (job != NULL) {
2401     /* look at the job object (maybe get the id) */
2402     papiJobFree(job);
2403 }
2404 ...
```

2405 **7.1.8 See Also**

2406 [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamOpen](#), [papiJobStreamWrite](#),
2407 [papiJobStreamClose](#), [papiJobFree](#)

2408 **7.2 papiJobSubmitByReference**

2409 **7.2.1 Description**

2410 Submits a print job having the specified attributes to the specified printer. This interface
2411 delays copying the specified print files as long as possible, ideally only "pulling" the files
2412 when the printer is actually printing the job (contrast to [papiJobSubmit](#)).
2413 Attributes of the print job may be passed in the `job_attributes` argument and/or in a job
2414 ticket (using the `job_ticket` argument). If both are specified, the attributes in the
2415 `job_attributes` list will be applied to the `job_ticket` attributes and the resulting attribute set
2416 will be used.

2417 **7.2.2 Semantics Reference**

2418 Print-URI in [RFC2911], section 3.2.2

2419 **7.2.3 Syntax**

```
2420 papi_status_t papiJobSubmitByReference(papi_service_t handle,
2421                                       char *printer_name,
2422                                       papi_attribute_t **job_attributes,
2423                                       papi_job_ticket_t *job_ticket,
```

2424 `char **file_names, papi_job_t *job);`

2425 **7.2.4 Inputs**

2426 **7.2.4.1 handle**

2427 Handle to the print service to use.

2428 **7.2.4.2 printer_name**

2429 Pointer to the name of the printer to which the job is to be submitted.

2430 **7.2.4.3 job_attributes**

2431 (optional) The list of attributes describing the job and how it is to be printed. If options are
2432 specified here and also in the job ticket data, the value specified here takes precedence. If
2433 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2434 job.

2435 **7.2.4.4 job_ticket**

2436 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2437 job ticket is used with the job. Whether the implementation passes both the attributes and
2438 the job ticket to the server/printer, or merges them to some print protocol or internal
2439 representation depends on the implementation.

2440 **7.2.4.5 file_names**

2441 NULL terminated list of pointers to names of files to print. If more than one file is
2442 specified, the files will be treated by the print system as separate "documents" for things
2443 like page breaks and separator sheets, but they will be scheduled and printed together as one
2444 job and the specified attributes will apply to all the files.

2445 These file names may contain absolute path names, relative path names or URIs
2446 ([RFC1738], [RFC2396]). The implementation SHOULD NOT copy the referenced data
2447 unless (or until) it is no longer feasible to maintain the reference. Feasibility limitations
2448 may arise out of security issues, name space issues, and/or protocol or printer limitations.
2449 Implementations MUST support the absolute path, relative path, and "file:" URI scheme.
2450 Use of other URI schemes could result in a PAPI_URI_SCHEME error, depending on the
2451 implementation.

2452 The semantics explained in the preceding paragraphs allows for flexibility in the PAPI
2453 implementation. For example: (1) PAPI on top of a local service to maintain the reference
2454 for the life of the job, if the local service supports it. (2) PAPI on top of IPP to send a
2455 reference when the server can access the referenced data and copy it when it is not
2456 accessible to the server. (3) PAPI on top of network printing protocols that don't support
2457 references to copy the data on the way out to the remote server.

2458 7.2.5 Outputs

2459 7.2.5.1 job

2460 The resulting job object representing the submitted job. The caller must deallocate this
2461 object using [papiJobFree\(\)](#) when finished using it.
2462

2463 7.2.6 Returns

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

2466 7.2.7 Example

```
2467 papi_status_t status;  
2468 papi_service_t handle = NULL;  
2469 papi_attribute_t **attrs = NULL;  
2470 papi_job_ticket_t *ticket = NULL;  
2471 char *files[] = { "/etc/motd", NULL };  
2472 papi_job_t job = NULL;  
2473 ...  
2474 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,  
2475                             PAPI_STRING, 1, "test job");  
2476 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,  
2477                              PAPI_INTEGER, 4);  
2478  
2479 status = papiJobSubmitByReference(handle, "printer", attrs, ticket,  
2480                                   files, &job);  
2481 papiAttributeListFree(attrs)  
2482 ...  
2483 if (job != NULL) {  
2484     /* look at the job object (maybe get the id) */  
2485     papiJobFree(job);  
2486 }  
2487 ...
```

2488 7.2.8 See Also

2489 [papiJobSubmit](#), [papiJobValidate](#), [papiJobStreamOpen](#), [papiJobStreamWrite](#),
2490 [papiJobStreamClose](#), [papiJobFree](#)

2491 7.3 *papiJobValidate*

2492 7.3.1 Description

2493 Validates the specified job attributes against the specified printer. This function can be used
2494 to validate the capability of a print object to accept a specific combination of attributes.
2495 Attributes of the print job may be passed in the job_attributes argument and/or in a job

2496 ticket (using the `job_ticket` argument). If both are specified, the attributes in the
2497 `job_attributes` list will be applied to the `job_ticket` attributes and the resulting attribute set
2498 will be used.

2499 **7.3.2 Semantics Reference**

2500 Validate-Job in [RFC2911], section 3.2.3

2501 **7.3.3 Syntax**

```
2502 papi_status_t papiJobValidate(papi_service_t handle, char *printer_name,  
2503                             papi_attribute_t **job_attributes,  
2504                             papi_job_ticket_t *job_ticket,  
2505                             char **file_names, papi_job_t *job );
```

2506 **7.3.4 Inputs**

2507 **7.3.4.1 *handle***

2508 Handle to the print service to use.

2509 **7.3.4.2 *printer_name***

2510 Pointer to the name of the printer to which the job is to be validated.

2511 **7.3.4.3 *job_attributes***

2512 (optional) The list of attributes describing the job and how it is to be printed. If options are
2513 specified here and also in the job ticket data, the value specified here takes precedence. If
2514 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2515 job.

2516 **7.3.4.4 *job_ticket***

2517 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2518 job ticket is used with the job. Whether the implementation passes both the attributes and
2519 the job ticket to the server/printer, or merges them to some print protocol or internal
2520 representation depends on the implementation.

2521 **7.3.4.5 *file_names***

2522 NULL terminated list of pointers to names of files to validate.

2523 **7.3.5 Outputs**

2524 **7.3.5.1 *job***

2525 The resulting job object representing the validated job. The caller must deallocate this

Chapter 7: Job API

2526 object using [papiJobFree\(\)](#) when finished using it.

2527

2528 7.3.6 Returns

2529 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2530 returned.

2531 7.3.7 Example

```
2532 papi_status_t status;  
2533 papi_service_t handle = NULL;  
2534 papi_attribute_t **attrs = NULL;  
2535 papi_job_ticket_t *ticket = NULL;  
2536 char *files[] = { "/etc/motd", NULL };  
2537 papi_job_t job = NULL;  
2538 ...  
2539 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,  
2540                             PAPI_STRING, 1, "test job");  
2541 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,  
2542                              PAPI_INTEGER, 4);  
2543 ...  
2544 status = papiJobValidate(handle, printer, attrs, ticket, files, &job);  
2545 papiAttributeListFree(attrs);  
2546 ...  
2547 if (job != NULL) {  
2548     papiJobFree(job);  
2549 }  
2550 ...
```

2551 7.3.8 See Also

2552 [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobStreamOpen](#), [papiJobStreamWrite](#),
2553 [papiJobStreamClose](#), [papiJobFree](#)

2554 7.4 *papiJobStreamOpen*

2555 7.4.1 Description

2556 Opens a print job and an associated stream of print data to be sent to the specified printer.
2557 After calling this function [papiJobStreamWrite](#) can be called (repeatedly) to write the print
2558 data to the stream, and then [papiJobStreamClose](#) is called to complete the submission of the
2559 print job.

2560 After this function is called successfully, [papiJobStreamClose](#) must eventually be called to
2561 close the stream (this includes all error paths).

2562 Attributes of the print job may be passed in the job_attributes argument and/or in a job
2563 ticket (using the job_ticket argument). If both are specified, the attributes in the
2564 job_attributes list will be applied to the job_ticket attributes and the resulting attribute set

2565 will be used.

2566 **7.4.2 Syntax**

```
2567 papi_status_t papiJobStreamOpen(papi_service_t handle, char *printer_name,  
2568                               papi_attribute_t **job_attributes,  
2569                               papi_job_ticket_t *job_ticket,  
2570                               papi_stream_t *stream);
```

2571 **7.4.3 Inputs**

2572 **7.4.3.1 handle**

2573 Handle to the print service to use.

2574 **7.4.3.2 printer_name**

2575 Pointer to the name of the printer to which the job is to be validated.

2576 **7.4.3.3 job_attributes**

2577 (optional) The list of attributes describing the job and how it is to be printed. If options are
2578 specified here and also in the job ticket data, the value specified here takes precedence. If
2579 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2580 job.

2581 **7.4.3.4 job_ticket**

2582 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2583 job ticket is used with the job. Whether the implementation passes both the attributes and
2584 the job ticket to the server/printer, or merges them to some print protocol or internal
2585 representation depends on the implementation.

2586 **7.4.4 Outputs**

2587 **7.4.4.1 stream**

2588 The resulting stream object to which print data can be written. The stream object will be
2589 deallocated when closed using [papiJobStreamClose\(\)](#).
2590

2591 **7.4.5 Returns**

2592 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2593 returned.

2594 7.4.6 Example

```
2595 papi_status_t status;
2596 papi_service_t handle = NULL;
2597 papi_attribute_t **attrs = NULL;
2598 papi_job_ticket_t *ticket = NULL;
2599 papi_job_t job = NULL;
2600 char buffer[4096];
2601 size_t buflen = 0;
2602 ...
2603 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
2604                             PAPI_STRING, 1, "test job");
2605 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
2606                              PAPI_INTEGER, 4);
2607 ...
2608 status = papiJobStreamOpen(handle, "printer", attrs, ticket, &stream);
2609 papiAttributeListFree(attrs);
2610 ...
2611 while (print_data_remaining) {
2612     status = papiJobStreamWrite(handle, stream, buffer, buflen);
2613 }
2614 ...
2615 status = papiJobStreamClose(handle, stream, &job);
2616 ...
2617 if (job != NULL) {
2618     ...
2619     papiJobFree(job);
2620 }
2621 ...
```

2622 7.4.7 See Also

2623 [papiJobStreamWrite](#), [papiJobStreamClose](#)

2624 7.5 *papiJobStreamWrite*

2625 7.5.1 Description

2626 Writes print data to the specified open job stream. The open job stream must have been
2627 obtained by a successful call to [papiJobStreamOpen](#)

2628 7.5.2 Syntax

```
2629 papi_status_t papiJobStreamWrite(papi_service_t handle, papi_stream_t stream,
2630                                 void *buffer, size_t buflen);
```

2631 **7.5.3 Inputs**

2632 **7.5.3.1 handle**

2633 Handle to the print service to use.

2634 **7.5.3.2 stream**

2635 The open stream object to which print data is written.

2636 **7.5.3.3 buffer**

2637 Pointer to the buffer of print data to write.

2638 **7.5.3.4 buflen**

2639 The number of bytes to write.

2640 **7.5.4 Outputs**

2641 none

2642 **7.5.5 Returns**

2643 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2644 returned.

2645 **7.5.6 Example**

2646 See `papiJobStreamOpen`

2647 **7.5.7 See Also**

2648 [papiJobStreamOpen](#), [papiJobStreamClose](#)

2649 **7.6 papiJobStreamClose**

2650 **7.6.1 Description**

2651 Closes the specified open job stream and completes submission of the job (if there were no
2652 previous errors returned from `papiJobSubmitWrite`). The open job stream must have been
2653 obtained by a successful call to `papiJobStreamOpen`.

2654 **7.6.2 Syntax**

```
2655 papi_status_t papiJobStreamClose(papi_service_t handle, papi_stream_t stream,  
2656                                papi_job_t *job);
```

2657 **7.6.3 Inputs**

2658 **7.6.3.1 handle**

2659 Handle to the print service to use.

2660 **7.6.3.2 stream**

2661 The open stream object to close.

2662 **7.6.4 Outputs**

2663 **7.6.4.1 Job**

2664 The resulting job object representing the submitted job. The caller must deallocate this
2665 object using [papiJobFree\(\)](#) when finished using it.

2666 **7.6.5 Returns**

2667 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2668 returned.

2669 **7.6.6 Example**

2670 See [papiJobStreamOpen](#)

2671 **7.6.7 See Also**

2672 [papiJobStreamOpen](#), [papiJobStreamWrite](#)

2673 **7.7 papiJobQuery**

2674 **7.7.1 Description**

2675 Queries some or all the attributes of the specified job object.

2676 **7.7.2 Semantics Reference**

2677 Get-Job-Attributes in [RFC2911], section 3.3.4

2678 **7.7.3 Syntax**

```
2679 papi_status_t papiJobQuery(papi_service_t handle, char* printer_name,  
2680                          int32_t job_id, char *requested_attrs[],  
2681                          papi_job_t *job);
```

2682 **7.7.4 Inputs**

2683 **7.7.4.1 handle**

2684 Handle to the print service to use.

2685 **7.7.4.2 printer_name**

2686 Pointer to the name or URI of the printer to which the job was submitted.

2687 **7.7.4.3 job_id**

2688 The ID number of the job to be queried.

2689 **7.7.4.4 requested_attrs**

2690 NULL terminated array of attributes to be queried. If NULL is passed then all available
2691 attributes are queried. (NOTE: The job may return more attributes than you requested. This
2692 is merely an advisory request that may reduce the amount of data returned if the
2693 printer/server supports it.)

2694 **7.7.5 Outputs**

2695 **7.7.5.1 job**

2696 The returned job object containing the requested attributes.

2697 **7.7.6 Returns**

2698 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2699 returned.

2700 **7.7.7 Example**

```
2701 papi_status_t status;  
2702 papi_service_t handle = NULL;  
2703 papi_job_t job = NULL;  
2704 char *job_attrs[] = {  
2705     "job-id", "job-name", "job-originating-user-name",  
2706     "job-state", "job-state-reasons", NULL };  
2707 ...  
2708 status = papiJobQuery(handle, "printer", job_id, job_attrs, &job);  
2709 ...  
2710 if (job != NULL) {  
2711     /* process the job */  
2712     ...  
2713     papiJobFree(job);  
2714 }  
2715 ...
```

2716 **7.7.8 See Also**

2717 [papiPrinterListJobs](#), [papiJobFree](#)

2718 **7.8 papiJobModify**

2719 **7.8.1 Description**

2720 Modifies some or all the attributes of the specified job object. Upon successful completion,
2721 the function will return a handle to an object representing the updated job.

2722 **7.8.2 Semantics Reference**

2723 Set-Job-Attributes in [RFC3380], section 4.2

2724 **7.8.3 Syntax**

```
2725 papi_status_t papiJobModify(papi_service_t handle, char* printer_name,  
2726                             int32_t job_id, papi_attribute_t **attrs,  
2727                             papi_job_t *job);
```

2728 **7.8.4 Inputs**

2729 **7.8.4.1 handle**

2730 Handle to the print service to use.

2731 **7.8.4.2 printer_name**

2732 Pointer to the name or URI of the printer to which the job was submitted.

2733 **7.8.4.3 job_id**

2734 The ID number of the job to be queried.

2735 **7.8.4.4 attrs**

2736 Attributes to be modified. Any attributes not specified are left unchanged.

2737 **7.8.5 Outputs**

2738 **7.8.5.1 job**

2739 The modified job object.

2740 **7.8.6 Returns**

2741 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2742 returned.

2743 7.8.7 Example

```
2744 papi_status_t status;
2745 papi_service_t handle = NULL;
2746 papi_job_t job = NULL;
2747 papi_attribute_t **attrs = NULL;
2748 ...
2749 papiAttributeListAddString(&attrs, PAPI_EXCL,
2750                             "job-name", "sample job");
2751 papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
2752                               "media", PAPI_DELETE);
2753 ...
2754 status = papiJobModify(handle, "printer", 12, attrs, &job);
2755 ...
2756 if (job != NULL) {
2757     /* process the job */
2758     ...
2759     papiJobFree(job);
2760 }
2761 ...
```

2762 7.8.8 See Also

2763 [papiJobFree](#)

2764 7.9 *papiJobCancel*

2765 7.9.1 Description

2766 Cancel the specified print job

2767 7.9.2 Semantics Reference

2768 Cancel Job in [RFC2911], section 3.3.3

2769 7.9.3 Syntax

```
2770 papi_status_t papiJobCancel(papi_service_t handle, char* printer_name,
2771                             int32_t job_id);
```

2772 7.9.4 Inputs

2773 7.9.4.1 *handle*

2774 Handle to the print service to use.

2775 7.9.4.2 *printer_name*

2776 Pointer to the name or URI of the printer to which the job was submitted.

2777 **7.9.4.3 *job_id***

2778 The ID number of the job to be canceled.

2779 **7.9.5 Outputs**

2780 none

2781 **7.9.6 Returns**

2782 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2783 returned.

2784 **7.9.7 Example**

```
2785 papi_status_t status;  
2786 papi_service_t handle = NULL;  
2787 ...  
2788 status = papiJobCancel(handle, "printer", 12);  
2789 ...
```

2790 **7.9.8 See Also**

2791 [papiPrinterPurgeJobs](#)

2792 **7.10 *papiJobHold***

2793 **7.10.1 Description**

2794 Hold the specified print job

2795 **7.10.2 Semantics Reference**

2796 Hold Job in [RFC2911], section 3.3.5

2797 **7.10.3 Syntax**

```
2798 papi_status_t papiJobHold(papi_service_t handle, char* printer_name,  
2799                          int32_t job_id);
```

2800 **7.10.4 Inputs**

2801 **7.10.4.1 *handle***

2802 Handle to the print service to use.

2803 **7.10.4.2 printer_name**

2804 Pointer to the name or URI of the printer to which the job was submitted.

2805 **7.10.4.3 job_id**

2806 The ID number of the job to be held.

2807 **7.10.5 Outputs**

2808 none

2809 **7.10.6 Returns**

2810 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2811 returned.

2812 **7.10.7 Example**

```
2813 papi_status_t status;  
2814 papi_service_t handle = NULL;  
2815 ...  
2816 status = papiJobHold(handle, "printer", 12);  
2817 ...
```

2818 **7.10.8 See Also**

2819 [papiJobRelease](#)

2820 **7.11 papiJobRelease**

2821 **7.11.1 Description**

2822 Release the specified print job

2823 **7.11.2 Semantics Reference**

2824 Release Job in [RFC2911], section 3.3.6

2825 **7.11.3 Syntax**

```
2826 papi_status_t papiJobRelease(papi_service_t handle, char* printer_name,  
2827                             int32_t job_id);
```

2828 **7.11.4 Inputs**

2829 **7.11.4.1 handle**

2830 Handle to the print service to use.

2831 **7.11.4.2 printer_name**

2832 Pointer to the name or URI of the printer to which the job was submitted.

2833 **7.11.4.3 job_id**

2834 The ID number of the job to be released.

2835 **7.11.5 Outputs**

2836 none

2837 **7.11.6 Returns**

2838 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2839 returned.

2840 **7.11.7 Example**

```
2841 papi_status_t status;  
2842 papi_service_t handle = NULL;  
2843 ...  
2844 status = papiJobRelease(handle, "printer", 12);  
2845 ...
```

2846 **7.11.8 See Also**

2847 [papiJobHold](#)

2848 **7.12 papiJobRestart**

2849 **7.12.1 Description**

2850 Restarts a job that was retained after processing. If and how a job is retained after
2851 processing is implementation-specific and is not covered by this API. This operation is
2852 optional and may not be supported by all printers/servers.

2853 **7.12.2 Semantics Reference**

2854 Restart Job in [RFC2911], section 3.3.7

2855 **7.12.3 Syntax**

```
2856 papi_status_t papiJobRestart(papi_service_t handle, char* printer_name,  
2857 int32_t job_id);
```

2858 **7.12.4 Inputs**

2859 **7.12.4.1 handle**

2860 Handle to the print service to use.

2861 **7.12.4.2 printer_name**

2862 Pointer to the name or URI of the printer to which the job was submitted.

2863 **7.12.4.3 job_id**

2864 The ID number of the job to be restart.

2865 **7.12.5 Outputs**

2866 none

2867 **7.12.6 Returns**

2868 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2869 returned.

2870 **7.12.7 Example**

```
2871 papi_status_t status;  
2872 papi_service_t handle = NULL;  
2873 ...  
2874 status = papiJobRestart(handle, "printer", 12);  
2875 ...
```

2876 **7.12.8 See Also**

2877 [papiJobHold](#), [papiJobRelease](#)

2878 **7.13 papiJobPromote**

2879 **7.13.1 Description**

2880 Promotes a job to the front of the queue so that it may be printed after any currently
2881 printing job completes. This operation is optional and may not be supported by all
2882 printers/servers.

2883 **7.13.2 Semantics Reference**

2884 Restart Job in [RFC2911], section 3.3.7

2885 **7.13.3 Syntax**

```
2886 papi_status_t papiJobPromote(papi_service_t handle,char* printer_name,  
2887                             int32_t job_id);
```

2888 **7.13.4 Inputs**

2889 **7.13.4.1 handle**

2890 Handle to the print service to use.

2891 **7.13.4.2 printer_name**

2892 Pointer to the name or URI of the printer to which the job was submitted.

2893 **7.13.4.3 job_id**

2894 The ID number of the job to be promoted.

2895 **7.13.5 Outputs**

2896 none

2897 **7.13.6 Returns**

2898 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2899 returned.

2900 **7.13.7 Example**

```
2901 papi_status_t status;  
2902 papi_service_t handle = NULL;  
2903 ...  
2904 status = papiJobPromote(handle, "printer", 12);  
2905 ...
```

2906 **7.13.8 See Also**

2907 [papiJobHold](#), [papiJobRelease](#)

2908 **7.14 *papiJobGetAttributeList***

2909 **7.14.1 Description**

2910 Get the attribute list associated with a job object.
2911 This function retrieves an attribute list from a job object returned in a previous call. Job
2912 objects are returned as a result of the operations performed by [papiPrinterListJobs](#),
2913 [papiJobQuery](#), [papiJobModify](#), [papiJobSubmit](#), [papiJobSubmitByReference](#),
2914 [papiJobValidate](#), and [papiJobStreamClose](#).

2915 **7.14.2 Syntax.**

```
2916 papi_attribute_t **papiJobGetAttributeList(papi_job_t job);
```

2917 **7.14.3 Inputs**

2918 **7.14.3.1 *job***

2919 Handle of the job object.

2920 **7.14.4 Outputs**

2921 none

2922 **7.14.5 Returns**

2923 The attribute list associated with the job object. The attribute list is deallocated when the
2924 containing job object is destroyed using [papiJobFree\(\)](#).

2925 **7.14.6 Example**

```
2926 papi_job_t job = NULL;  
2927 papi_attribute_list **attrs = NULL;  
2928 ...  
2929 attrs = papiJobGetAttributeList(job);  
2930 ...  
2931 papiJobFree(job);
```

2933 **7.14.7 See Also**

2934 [papiPrinterListJobs](#), [papiJobQuery](#), [papiJobModify](#), [papiJobSubmit](#),
2935 [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamClose](#)

2936 **7.15 *papiJobGetPrinterName***

2937 **7.15.1 Description**

2938 Get the printer name associated with a job object.

2939 **7.15.2 Syntax.**

```
2940 char *papiJobGetPrinterName(papi_job_t job);
```

2941 **7.15.3 Inputs**

2942 **7.15.3.1 *job***

2943 Handle of the job object.

2944 **7.15.4 Outputs**

2945 none

2946 **7.15.5 Returns**

2947 Pointer to the printer name associated with the job object. The resulting string is
2948 deallocated when the containing job object is destroyed using [papiJobFree\(\)](#).

2949 **7.15.6 Example**

```
2950 char *printer = NULL;  
2951 papi_job_t job = NULL;  
2952 ...  
2953 printer = papiJobGetPrinterName(job);  
2954 ...  
2955 papiJobFree(job);
```

2957 **7.15.7 See Also**

2958 [papiJobGetAttributeList](#)

2959 **7.16 *papiJobGetId***

2960 **7.16.1 Description**

2961 Get the job ID associated with a job object.

2962 **7.16.2 Syntax.**

```
2963 int32_t papiJobGetId(papi_job_t job);
```

2964 **7.16.3 Inputs**

2965 **7.16.3.1 job**

2966 Handle of the job object.

2967 **7.16.4 Outputs**

2968 none

2969 **7.16.5 Returns**

2970 The job id associated with the job object.

2971 **7.16.6 Example**

```
2972 papi_job_t job = NULL;
2973 int32_t id;
2974 ...
2975 id = papiJobGetId(job);
2976 ...
2977 papiJobFree(job);
```

2978 **7.16.7 See Also**

2979 [papiJobGetAttributeList](#)

2980 **7.17 papiJobGetJobTicket**

2981 **7.17.1 Description**

2982 Get the job ticket associated with a job object. The job ticket is deallocated when the
2983 containing job object is destroyed using [papiJobFree\(\)](#).

2984 **7.17.2 Syntax**

```
2985 papi_job_ticket_t *papiJobGetJobTicket(papi_job_t job);
```

2986 **7.17.3 Inputs**

2987 **7.17.3.1 job**

2988 Handle of the job object.

2989 **7.17.4 Outputs**

2990 none

2991 **7.17.5 Returns**

2992 Pointer to the job ticket associated with the job object.

2993 **7.17.6 Example**

```
2994 papi_job_t job = NULL;  
2995 papi_job_ticket_t *ticket;  
2996 ...  
2997 ticket = papiJobGetJobTicket(job);  
2998 ...  
2999 papiJobFree(job);
```

3000 **7.17.7 See Also**

3001 [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamOpen](#)

3002 **7.18 *papiJobFree***

3003 **7.18.1 Description**

3004 Free a job object.

3005 **7.18.2 Syntax**

```
3006 void papiJobFree(papi_job_t job);
```

3007 **7.18.3 Inputs**

3008 **7.18.3.1 *Job***

3009 Handle of the job object to free.

3010 **7.18.4 Outputs**

3011 none

3012 **7.18.5 Returns**

3013 none

3014 **7.18.6 Example**

```
3015 papi_job_t job = NULL;
```

```
3016 ...
3017 papiJobFree (job);
```

3018 **7.18.7 See Also**

3019 [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamClose](#),
3020 [papiJobQuery](#), [papiJobModify](#)

3021 **7.19 papiJobListFree**

3022 **7.19.1 Description**

3023 Free a job list.

3024 **7.19.2 Syntax**

```
3025 void papiJobListFree(papi_job_t *job);
```

3026 **7.19.3 Inputs**

3027 **7.19.3.1 Job**

3028 Handle of the job list to free.

3029 **7.19.4 Outputs**

3030 none

3031 **7.19.5 Returns**

3032 none

3033 **7.19.6 Example**

```
3034 papi_job_t *jobs = NULL;
3035 ...
3036 papiJobListFree (jobs);
```

3037 **7.19.7 See Also**

3038 [papiPrinterListJobs](#)

3039 **Chapter 8: Miscellaneous API**

3040 **8.1 *papiStatusString***

3041 **8.1.1 Description**

3042 Get a status string for the specified `papi_status_t`. The status message returned from this
3043 function may be less detailed than the status message returned from
3044 [papiServiceGetStatusMessage](#) (if the print service supports returning more detailed error
3045 messages)

3046 **8.1.2 Syntax**

```
3047 char *papiStatusString(papi_status_t status);
```

3048 **8.1.3 Inputs**

3049 **8.1.3.1 *status***

3050 The status value to convert to a status string

3051 **8.1.4 Outputs**

3052 none

3053 **8.1.5 Returns**

3054 The returned string provides a (potentially localized) human readable message representing
3055 the status provided. The return value should not be deallocated by the caller.

3056 **8.1.6 Example**

```
3057 papi_status_t status;  
3058 char *message;  
3059 ...  
3060 message = papiServiceGetStatusMessage(handle);  
3061 ...
```

3062 **8.1.7 See Also**

3063 [PapiServiceGetStatusMessage](#)

3064 **8.2 *papiLibrarySupportedCalls***

3065 **8.2.1 Description**

3066 The `papiLibrarySupportedCalls()` function can be called to request a list of API functions

3067 that are supported in the implementation. Support for a function means that the
3068 implementation of that function is not a stub that simply returns
3069 PAPI_OPERATION_NOT_SUPPORTED

3070 **8.2.2 Syntax**

```
3071 char **papiLibrarySupportedCalls();
```

3072 **8.2.3 Inputs**

3073 none

3074 **8.2.4 Outputs**

3075 none

3076 **8.2.5 Returns**

3077 A NULL terminated list of supported function names. This list should not be deallocated
3078 by the caller.

3079 **8.2.6 Example**

```
3080 papi_service_t handle = NULL;  
3081 char **calls;  
3082 ...  
3083 calls = papiLibrarySupportedCalls(handle);  
3084 ...
```

3085 **8.2.7 See Also**

3086 Conformance Table

3087 **8.3 *papiLibrarySupportedCall***

3088 **8.3.1 Description**

3089 The papiLibrarySupportedCalls() function can be called to request a list of API functions
3090 that are supported in the implementation. Support for a function means that the
3091 implementation of that function is not a stub that simply returns
3092 PAPI_OPERATION_NOT_SUPPORTED

3093 **8.3.2 Syntax**

```
3094 char papiLibrarySupportedCall(char *name);
```

3095 **8.3.3 Inputs**

3096 **8.3.3.1 name**

3097 The name of the function that is being asked about

3098 **8.3.4 Outputs**

3099 none

3100 **8.3.5 Returns**

3101 A return of PAPI_TRUE indicates that the named function is supported by the API
3102 implementation. A return of PAPI_FALSE indicates that the the named function is not
3103 supported by the API implementation.

3104 **8.3.6 Example**

```
3105 papi_service_t handle = NULL;  
3106 char supported;  
3107 ...  
3108 supported = papiLibrarySupportedCall(handle, "papiJobQuery");  
3109 ...
```

3110 **8.3.7 See Also**

3111 [ConformanceProfiles](#)

3112 **Chapter 9: Capabilities**

3113 **9.1 Introduction**

3114 In the context of this document, printer capabilities refers to information about the features,
3115 options, limitation, etc. of a print device (either an actual device or an abstract device which
3116 may represent a group or pool of actual devices). This includes such information as:

- 3117 • Does the printer support color printing?
- 3118 • At what resolution(s) can the printer print?
- 3119 • What input trays are present?
- 3120 • What size media is loaded in each tray?
- 3121 • Which trays are manual-feed and which are auto-feed?
- 3122 • Can the printer print duplex output?
- 3123 • What is the printable area on each of the loaded media?
- 3124 • What output bins are present?
- 3125 • What finishing (staple, punch, etc.) does the printer support?
- 3126 • What combinations of features are not allowed together?
- 3127 • What features should be presented on the print user interface?
- 3128 • ... and many others...

3129 The uses of printer capabilities by applications include:

3130 To control how to display print options in a print UI dialog. Examples:

- 3131 • What values to put in the binselection pull-down lists
- 3132 • Whether or not to gray-out the duplex option when a particular output bin
3133 has been selected.
- 3134 • Whether or not to display a color vs. black and white selection

3135 To Control how the printdata stream is generated. Examples:

- 3136 • How large an image to draw to fill the printable area.
- 3137 • How much to shift the image if “3-hole punch” finishing has been
3138 selected.
- 3139 • How to request that the printer print on paper from the manual envelope
3140 feeder

3141 To do job validation and printer selection. Examples:

- 3142 • Can I print this job with these options on this printer?

- 3143 • Find a printer which can print this job with these options.

3144 **9.2 Objectives**

3145 This section attempts to describe the objectives of the PAPI printer capabilities support. It
3146 is important to understand these objectives in order to understand why the support is
3147 structured the way that it is.

3148 **9.2.1 Standard printer capabilities API**

3149 There is no standard API which a Linux application can use to retrieve printer capabilities
3150 regardless of the device, driver, and print server being used. This makes it very difficult for
3151 application writers to support generating print data without writing multiple versions of the
3152 print logic or without tying the application to very specific print system environments. This
3153 specification provides the standard API, making applications which use it independent of
3154 the underlying print system.

3155 **9.2.2 Independent of underlying source of capabilities**

3156 The capabilities information returned to the application may come from one of a variety of
3157 sources or combination of sources. The data retrieved from these sources may be
3158 represented in a variety of formats, including:

- 3159 • PPD files
- 3160 • UPDF database
- 3161 • SNMP queries
- 3162 • Device drivers

3163 The API defined here hides these differences so that the application is independent of data
3164 source and format used.

3165 **9.2.3 Support returning information in context**

3166 The API supports a means for requesting capabilities information in the context of a
3167 particular set of job options. For example, set of printer capabilities can be queried given
3168 that medium and color/black-and-white selections have already been made.

3169 **9.2.4 Support returning constraints**

3170 The API must support a means for returning constraints on printer capabilities. This allows
3171 applications to not submit job with disallowed combinations of options, and to display
3172 better print job dialogs (gray-out potentially conflicting options, highlight conflicting
3173 options that have been selected, display an error message when invalid combinations are
3174 submitted, etc.).

3175 The constraints returned should allow some level of “boolean logic”, including

3176 negation, to simplify the information returned. For example, to not allow doing finishing
3177 when transparencies are selected as the medium, it would be preferable if the constraints
3178 could express “(type – transparency) AND (finishing NOT = none)” instead of having to
3179 list a combination of “(type = transparency)” with every possible finishing value other than
3180 “none”.

3181 **9.2.5 Support returning display hints**

3182 The API should support a means of returning “display hints”. This is information that the
3183 application can use to display print options in a print dialog that is easy to use. For
3184 example, returning information about which options should be displayed on the “main
3185 window”, which should be displayed in an “advanced” dialog, and which should not be
3186 displayed at all.

3187 **9.2.6 Support logically grouping features**

3188 The API should support a means of returning logical groupings of printer features. This is
3189 information about combinations of lower-level features that can be displayed and selected
3190 as a group to make the user interface easier to use. For example, a group of features called
3191 “black-and-white-draft” could include a logical setting of the color, resolution, and print
3192 density options.

3193 The feature group support should be an open, extendible way for printer vendors and print
3194 administrators to express logical and commonly used groupings of print options that make it
3195 easier for end-users to take advantage of lower-level printer features. They should not be
3196 used to blindly list all possible combinations of a set of options, whether or not all the
3197 combinations make sense.

3198 **9.3 Interfaces**

3199 **9.3.1 Query Functionality**

3200 The API used by the application to retrieve printer capabilities is the [papiPrinterQuery](#)
3201 function. See the description of that function for further details.

3202 **9.3.2 Capability Attributes**

3203 In addition to the xxx-supported attributes defined by the IPP standard [RFC2911], this
3204 section defines new attributes needed to satisfy the objectives described above.

3205 **9.3.2.1 Job-constraints-col (1 setOf collection)**

3206 Constraints are expressed in the printer object's job-constraints-col attribute. This attribute
3207 is multi-valued with each value having collection syntax. Each value is, in fact, an attribute
3208 list that represents one combination of job attributes/values which are not allowed for that
3209 printer. If an attribute in the collection does not have a value, then all values of that
3210 attribute are disallowed in this combination.

Chapter 9: Capabilities

3211 The set of values associated with job-constraints-col represents the complete set of job
3212 attribute constraints associated with the containing printer object.

3213 The attribute values in job-constraints-col may also be in range syntax, if the corresponding
3214 job attribute has integer syntax. This represents the included (or excluded, if the attribute is
3215 named in job-constraints-inverted) range of values of that attribute within that constraint.

3216 **9.3.2.2 Job-constraints-inverted (1setOf type2 keyword)**

3217 The job-constraints-inverted attribute lists the names of other attributes in the current job-
3218 constraints-col value whose comparison logic must be inverted. That is, the values of
3219 named attributes are to be excluded (“not equal to” values) from the constraint. If an
3220 attribute name is not included in the job-constraints-inverted attribute, then that attribute's
3221 values are to be included (“equal to” values) in the constraint.

3222 You can think of the each attribute in a job-constraints-cols value as AND-ed together to
3223 express a disallowed combination of options: “(attr1 == values) AND (attr2 == values)
3224 AND ...”. The job constraints-inverted attribute lists those attribute/value comparisons
3225 which are to be “!=” instead of “==”.

3226 **9.3.3 Example**

3227 Here is an example of how the job-constraints-col attribute can be used to express various
3228 printer constraints. The example is expressed in pseudo-code with curly brackets enclosing
3229 each collection value and attributes within each collection are shown on separate lines with
3230 commas separating the values (this is the PAPI text encoding format documented in
3231 [Chapter 11: Attribute List Text Representation](#), with the addition of not-legal-syntax
3232 comments in “/* ... */” to help describe the examples):

```
3233 job-constraints-col = {  
3234     /*  
3235     * Constraint: no high print quality with 240 dpi resolution  
3236     * (print-quality == high) AND (printer-resolution == 240dpi)  
3237     */  
3238     {  
3239         print-quality = high  
3240         printer -resolution = 240dpi  
3241     },  
3242     /*  
3243     * Constraint: no transparency with duplex  
3244     * (sides != one-sided) AND (media – transparency)  
3245     */  
3246     {  
3247         job-constraints-inverted = sides  
3248         sides = one-sided  
3249     }
```

```

3250     media = transparency
3251 },
3252
3253 /*
3254  * Constraint: no finishing with heavy-stock media
3255  * (finishings != none) AND (media == heavy-stock)
3256  /
3257 {
3258     job-constraints-inverted = finishing
3259     finishings = none
3260     media = heavy-stock
3261 },
3262
3263 /*
3264  * Constraint: no duplex printing of A4 paper in landscape
3265  * (sides != one-sided) AND (media == A4) AND
3266  * (orientation-requested == landscape)
3267  /
3268 {
3269     job-constraints-inverted = sides
3270     sides = one-sided
3271     media = A4
3272     orientation-requested = landscape
3273 },
3274
3275 /*
3276  * Constraint: no duplex printing of COM-10 envelopes
3277  * (sides != one-sided) AND (media == envelope) AND
3278  * (media-size == com10)
3279  /
3280 {
3281     job-constraints-inverted = sides
3282     sides = one-sided
3283     media = envelope
3284     media-size = com10
3285 },
3286
3287 /*
3288  * Constraint: no stapling of greater than 50 sheets
3289  * (finishings == staple) AND (job-media-sheets > 50)
3290  */
3291 {
3292     job-constraints-inverted = job-media-sheets

```

Chapter 9: Capabilities

```
3293     finishings = staple
3294     job-media-sheets = 1-50
3295 }
3296 };
```

3297 **9.3.4 Validation Function**

3298 The API used by the application to validate print job attributes against printer capabilities is
3299 the [papiJobValidate](#) function. See the description of that function for further details.

3300 **Chapter 10: Attributes**

3301 For a summary of the IPP attributes which can be used with the PAPI interface, see:
3302 <ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf>

3303 **10.1 Extension Attributes**

3304 The following attributes are not currently defined by IPP, but may be used with this API.

3305 **10.1.1 Job-ticket-formats-supported**

3306 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats that are
3307 supported by the printer. If this attribute is not present, it is assumed that the printer does
3308 not support any job ticket formats

3309 **10.1.2 media-margins**

3310 (1setOf integer) The media-margins attribute defines the printable margins for the current
3311 printer object and consists of exactly 4 or 8 ordered integers. Each group of 4 integers
3312 represent the minimum distance from the top, right, bottom, and left edges of the media in
3313 100ths of millimeters.

3314 If 4 integers are provided, the margins are the same for the front and back sides of the
3315 media when producing duplexed output. If 8 integers are provided, the first 4 integers
3316 represent the margins for the front side and the last 4 integers represent the margins for the
3317 back side of the media.

3318 Currently the margin values only represent the minimum margins that can be used with all
3319 sizes and types of media. Future versions of the PAPI specification may define an interface
3320 for getting the margin values for specific combinations of job template attributes.

3321 **10.2 Required Job Attributes**

3322 The following job attributes must be supported to comply with this API standard. These
3323 attributes may be supported by the underlying print server directly, or they may be mapped
3324 by the PAPI library.

- 3325 • job-id
- 3326 • job-name
- 3327 • job-originating-user-name
- 3328 • job-printer-uri
- 3329 • job-state
- 3330 • job-state-reasons
- 3331 • job-uri

Chapter 10: Attributes

- 3332 • time-at-creation
- 3333 • time-at-processing
- 3334 • time-at-completed

3335 **10.3 Required Printer Attributes**

3336 The following printer attributes must be supported to comply with this API standard. These
3337 attributes may be supported by the underlying print server directly, or they may be mapped
3338 by the PAPI library.

- 3339 • charset-configured
- 3340 • charset-supported
- 3341 • compression-supported
- 3342 • document-format-default
- 3343 • document-format-supported
- 3344 • generated-natural-language-supported
- 3345 • natural-language-configured
- 3346 • operations-supported
- 3347 • pdl-override-supported
- 3348 • printer-is-accepting-jobs
- 3349 • printer-name
- 3350 • printer-state
- 3351 • printer-state-reasons
- 3352 • printer-up-time
- 3353 • printer-uri-supported
- 3354 • queued-job-count
- 3355 • uri-authentication-supported
- 3356 • uri-security-supported

3357 **10.4 IPP Attribute Type Mapping**

3358 The following table maps IPP to PAPI attribute value types:

<i>IPP Type</i>	<i>PAPI Type</i>
boolean	PAPI_BOOLEAN
charset	PAPI_STRING
collection	PAPI_COLLECTION
dateTime	PAPI_DATETIME
enum	PAPI_INTEGER (with C enum values)
integer	PAPI_INTEGER
keyword	PAPI_STRING
mimeMediaType	PAPI_STRING
name	PAPI_STRING
naturalLanguage	PAPI_STRING
octetString	not yet supported
rangeOfInteger	PAPI_RANGE
resolution	PAPI_RESOLUTION
text	PAPI_STRING
uri	PAPI_STRING
uriScheme	PAPI_STRING
1setOf X	C array
OOB (unused, delete, unsupported, etc.)	PAPI_METADATA (with enum value)

3359

3360 **Chapter 11: Attribute List Text Representation**3361 **11.1 ABNF Definition**

3362 The following ABNF definition [RFC2234] describes the syntax of PAPI attributes
 3363 encoded as text options:

```

3364 OPTION-STRING = [OPTION] *(1*WC OPTION) *WC
3365
3366 OPTION          = TRUEOPTION / FALSEOPTION / VALUEOPTION
3367
3368 TRUEOPTION      = NAME
3369
3370 FALSEOPTION     = "no" NAME
3371
3372 VALUEOPTION     = NAME "=" VALUE *( "," VALUE )
3373
3374 NAME           = 1*NAMECHAR
3375
3376 NAMECHAR       = DIGIT / ALPHA / "-" / "_" / "."
3377
3378 VALUE          = BOOLVALUE / COLVALUE / DATEVALUE / NUMBERTVALUE /
3379 QUOTEDVALUE /
3380 RANGEVALUE / RESVALUE / STRINGVALUE
3381
3382 BOOLVALUE      = "yes" / "no" / "true" / "false"
3383
3384 COLVALUE       = "{" OPTION-STRING "}"
3385
3386 DATEVALUE      = HOUR MINUTE [ SECOND ] / YEAR MONTH DAY /
3387 YEAR MONTH DAY HOUR MINUTE [ SECOND ]
3388
3389 YEAR           = 4DIGIT
3390
3391 MONTH         = "0" %x31-39 / "10" / "11" / "12"
3392
3393 DAY           = %x30-32 DIGIT / "1" DIGIT / "2" DIGIT / "30" / "31"
3394
3395 HOUR          = %x30-31 DIGIT / "1" DIGIT / "20" / "21" / "22" / "23"
3396
3397 MINUTE        = %x30-35 DIGIT
3398
3399 SECOND        = %x30-35 DIGIT
3400
3401 NUMBERTVALUE  = 1*DIGIT / "-" 1*DIGIT / "+" 1*DIGIT
3402
3403 QUOTEDVALUE   = DQUOTE *QUOTEDCHAR DQUOTE / SQUOTE *QUOTEDCHAR SQUOTE
3404
3405 QUOTEDCHAR    = %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
3406 %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
3407 %x5D-7E / %xA0-FF

```



```

3408
3409 OCTALDIGIT      = %x30-37
3410
3411 RANGEVALUE     = 1*DIGIT "-" 1*DIGIT
3412
3413 RESVALUE       = 1*DIGIT [ "x" 1*DIGIT ] ("dpi" / "dpc")
3414
3415 STRINGVALUE    = 1*STRINGCHAR
3416
3417 STRINGCHAR     = %x5C %x20 / %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
3418                %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
3419                %x5D-7E / %xA0-FF
3420
3421 SQUOTE        = %x27
3422
3423 WC            = %x09 / %x0A / %x20

```

3424 **11.2 Examples**

3425 The following example strings illustrate the format of text options:

3426 **11.2.1 Boolean Attributes**

```

3427 foo
3428 nofoo
3429 foo=false
3430 foo=true
3431 foo=no
3432 foo=yes
3433

```

3434

3435 **11.2.2 Collection Attributes**

```

3436 media-col={media-size={x-dimension=123 y-dimension=456}}
3437

```

3438

3439 **11.2.3 Integer Attributes**

```

3440 copies=123
3441 hue=-123
3442

```

3443

3444 **11.2.4 String Attributes**

```
3445 job-sheets=standard
3446 job-sheets=standard,standard
3447 media=na-custom-foo.8000-10000
3448 job-name=John\'s\ Really\040Nice\ Document
3449
```

3450

3451 **11.2.5 String Attributes (quoted)**

```
3452 job-name="John\'s Really Nice Document"
3453 document-name='Another \'Word\042 document.doc'
3454
```

3455

3456 **11.2.6 Range Attributes**

```
3457 page-ranges=1-5
3458 page-ranges=1-2,5-6,101-120
3459
```

3460

3461 **11.2.7 Date Attributes**

```
3462 job-hold-until-datetime=1234
3463 job-hold-until-datetime=123456
3464 job-hold-until-datetime=20020904
3465 job-hold-until-datetime=200209041234
3466 job-hold-until-datetime=20020904123456
3467
```

3468

3469 **11.2.8 Resolution Attributes**

```
3470 resolution=360dpi
3471 resolution=720x360dpi
3472 resolution=1000dpc
3473
```

3474

3475 **11.2.9 Multiple Attributes**

```
3476 job-sheets=standard page-ranges=1-2,5-6,101-120 resolution=360dpi
```

3477 Chapter 12: Conformance

3478 There are some cases where it may not be necessary or even desirable to implement the
3479 interfaces defined in this specification in their entirety. This section describes which
3480 elements of the interfaces must be implemented and defines sets of interfaces that may be
3481 implemented. The sets of interfaces that may be implemented define various levels of
3482 conformance. Conformance to a particular level may only be claimed by an
3483 implementation if and only if all of the interfaces defined in that level are implemented as
3484 described in their associated section of this document. These implementations may only
3485 return PAPI_OPERATION_NOT_SUPPORTED if and only if the underlying support has
3486 been administratively disabled. Regardless of conformance level claimed by an
3487 implementation, the header file for every implementation must be complete. That is to say
3488 that it must include a complete set of type definitions, enumeration and function prototypes.

3489 12.1 Query Profile

3490 The Query Profile is defined to provide querying functionality. A PAPI implementation
3491 conforming to the Query Profile must provide code for all functions defined in the PAPI
3492 and must support all of the definitions in the “papi.h” C header file. For each function
3493 defined in the PAPI specification, a conforming implementation must either perform the
3494 requested function or return the PAPI_OPERATION_NOT_SUPPORTED status code (see
3495 section 3.8). The PAPI_OPERATION_NOT_SUPPORTED status code indicates either (1)
3496 that the PAPI implementation doesn’t provide any support for the function, i.e., the function
3497 is stubbed out, or (2), the PAPI implementation does provide *code support* for the function,
3498 but the Printer or Print system selected by the application does not support the
3499 corresponding function.

3500
3501 lists the functions and attributes that a PAPI implementation is REQUIRED to provide
3502 *code support* in order to claim conformance to the Query Profile. The blank entries are
3503 OPTIONAL for a PAPI implementation to support.

3504 12.2 Job Submission Profile

3505 The Job Submission Profile is defined to provide the job submission functionality and is a
3506 superset of the Querying Profile. lists the functions and attributes that a PAPI
3507 implementation is REQUIRED to provide *code support* in order to claim conformance to
3508 the Job Submission Profile. The blank entries are OPTIONAL for a PAPI implementation
3509 to support.

3510 12.3 Conformance Table

<i>PAPI Functions & Attributes</i>	<i>Query Profile</i>	<i>Job Submission Profile</i>
Chapter3: Common Structures	All Structures	All Structures

Chapter 12: Conformance

<i>PAPI Functions & Attributes</i>	<i>Query Profile</i>	<i>Job Submission Profile</i>
Chapter4: Attributes API		
4.1 papiAttributeListAddValue	REQUIRED	REQUIRED
4.2 papiAttributeListAddString	REQUIRED	REQUIRED
4.3 papiAttributeListAddInteger	REQUIRED	REQUIRED
4.4 papiAttributeListAddBoolean	REQUIRED	REQUIRED
4.5 papiAttributeListAddRange	REQUIRED	REQUIRED
4.6 papiAttributeListAddResolution	REQUIRED	REQUIRED
4.7 papiAttributeListAddDatetime	REQUIRED	REQUIRED
4.8 papiAttributeListAddCollection	REQUIRED	REQUIRED
4.9 papiAttributeListDelete	REQUIRED	REQUIRED
4.10 papiAttributeListGetValue	REQUIRED	REQUIRED
4.11 papiAttributeListGetString	REQUIRED	REQUIRED
4.12 papiAttributeListGetInteger	REQUIRED	REQUIRED
4.13 papiAttributeListGetBoolean	REQUIRED	REQUIRED
4.14 papiAttributeListGetRange	REQUIRED	REQUIRED
4.15 papiAttributeListGetResolution	REQUIRED	REQUIRED
4.16 papiAttributeListGetDatetime	REQUIRED	REQUIRED
4.17 papiAttributeListGetCollection	REQUIRED	REQUIRED
4.18 papiAttributeListFree	REQUIRED	REQUIRED
4.19 papiAttributeListFind	REQUIRED	REQUIRED
4.20 papiAttributeListGetNext	REQUIRED	REQUIRED
4.21 papiAttributeListFromString		
4.22 papiAttributeListToString		
Chapter5: Service API	All Functions	All Functions
Chapter6: Printer API		
6.2 papiPrintersList	REQUIRED	REQUIRED
6.3 papiPrinterQuery	REQUIRED	REQUIRED
6.4 papiPrinterModify		
6.5 papiPrinterPause		

<i>PAPI Functions & Attributes</i>	<i>Query Profile</i>	<i>Job Submission Profile</i>
6.6 papiPrinterResume		
6.7 papiPrinterPurgeJobs		
6.8 papiPrinterListJobs	REQUIRED	REQUIRED
6.9 papiPrinterGetAttributeList	REQUIRED	REQUIRED
6.10 papiPrinterFree	REQUIRED	REQUIRED
6.11 papiPrinterListFree	REQUIRED	REQUIRED
Chapter7: Job API		
7.1 papiJobSubmit		REQUIRED
7.2 papiJobSubmitByReference		REQUIRED
7.3 papiJobValidate		
7.4 papiJobStreamOpen		REQUIRED
7.5 papiJobStreamWrite		REQUIRED
7.6 papiJobStreamClose		REQUIRED
7.7 papiJobQuery	REQUIRED	REQUIRED
7.8 papiJobModify		REQUIRED
7.9 papiJobCancel		REQUIRED
7.10 papiJobHold		REQUIRED
7.11 papiJobRelease		REQUIRED
7.12 papiJobRestart		REQUIRED
7.13 papiJobGetAttributeList		REQUIRED
7.14 papiJobGetPrinterName		REQUIRED
7.15 papiJobGetId		REQUIRED
7.16 papiJobGetJobTicket		
7.17 papiJobFree		REQUIRED
7.18 papiJobListFree		REQUIRED
Chapter8: Miscellaneous API		
8.1 papiStatusString		REQUIRED
8.2 papiLibrarySupportedCalls		REQUIRED
8.3 papiLibrarySupportedCall		REQUIRED

Chapter 12: Conformance

<i>PAPI Functions & Attributes</i>	<i>Query Profile</i>	<i>Job Submission Profile</i>
Chapter9: Attributes		
9.1.1 Job-ticket-formats-supported	REQUIRED	REQUIRED
9.1.2 media-margins	REQUIRED	REQUIRED
9.2 Required Job Attributes	REQUIRED	REQUIRED
9.3 Required Printer Attributes	REQUIRED	REQUIRED
9.4 IPP Attribute Type Mapping	REQUIRED	REQUIRED

3511

3512 **Chapter 13: Sample Code**

3513 Sample implementations of this specification and client applications built upon it can be
3514 found at <http://sf.net/projects/OpenPrinting/> While the implemenations and clients
3515 applications found there are intended to be true to the spec, they are not authoritative. This
3516 document is the athoritavie definition of the Free Standard Group Open Standard Print API
3517 (PAPI).

3518 **Chapter 14: References**

3519 **14.1 Internet Printing Protocol (IPP)**

3520 IETF RFCs can be obtained from "<http://www.rfc-editor.org/rfcsearch.html>". Other IPP
3521 documents can be obtained from "<http://www.pwg.org/ipp/index.html>" and
3522 "ftp://ftp.pwg.org/pub/pwg/ipp/new_XXX/".

[RFC2911] T. Hastings R. Herriot R. deBry S. Isaacson and P. Powell August 1998
Internet Printing Protocol/1.1: Model and Semantics (Obsoletes 2566)
[RFC3196] T. Hastings H. Holst C. Kugler C. Manros and P. Zehler November 2001
Internet Printing Protocol/1.1: Implementor's Guide
[RFC3380] T. Hastings R. Herriot C. Kugler and H. Lewis September 2002 Internet
Printing Protocol (IPP): Job and Printer Set Operations
[RFC3381] T. Hastings H. Lewis and R. Bergman September 2002 Internet Printing
Protocol (IPP): Job Progress Attributes
[RFC3382] R. deBry T. Hastings R. Herriot K. Ocke and P. Zehler September 2002
Internet Printing Protocol (IPP): The 'collection' attribute syntax
[5100.2] T. Hastings and R. Bergman IEEE-ISTO 5100.2 February 2001 Internet Printing
Protocol (IPP): output-bin attribute extension
[5100.3] T. Hastings and K. Ocke IEEE-ISTO 5100.3 February 2001 Internet Printing
Protocol (IPP): Production Printing Attributes
[5100.4] R. Herriot and K. Ocke IEEE-ISTO 5100.4 February 2001 Internet Printing
Protocol (IPP): Override Attributes for Documents and Pages
[5101.1] T. Hastings and D. Fullman IEEE-ISTO 5101.1 February 2001 Internet Printing
Protocol (IPP): finishings attribute values extension
[ops-set2] C. Kugler T. Hastings and H. Lewis July 2001 Internet Printing Protocol (IPP):
Job and Printer Administrative Operations

3523 **14.2 Job Ticket**

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

3524 **14.3 Printer Working Group (PWG)**

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

3525 **14.4 Other**

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

- 3526 **Chapter 15: Change History**
- 3527 **15.1 Version 1.0 (July 14, 2005)**
- 3528 Addressed last call comments:
- 3529 Changed papiAttributeListAdd to papiAttributeListAddValue
- 3530 fixed cut/paste errors in 4.10.6 and 6.9.3
- 3531 **15.2 Version 1.0 (May 9, 2005)**
- 3532 Added note about interfaces to be covered in the next release.
- 3533 **15.3 Version 0.92 (January 12, 2005).**
- 3534 Added administrative operations: papiPrinterAdd, papiPrinterRemove, papiPrinter Enable,
- 3535 papaPrinterDisable, papiJobPromote.
- 3536 **15.4 Version 0.91 (January 28, 2004).**
- 3537 Pruned several example code excerpts to the essential information required to get a better
- 3538 understanding of the various calls.
- 3539 Added/modified introductory text for Attribute, Service, Printer, and Job API chapters.
- 3540 Added papi_metadata_t type/support for various OOB IPP types that we need to support.
- 3541 Converted from SGML to OpenOffice to be able to use versioning, change bars, line
- 3542 number, ... (will begin using versioning and change bars after this release)
- 3543 Added numerous cross references.
- 3544 Added papiLibrarySupportedCall() and papiLibrarySupportedCalls(). To enumerate/verify
- 3545 actual support for a function in the library
- 3546 Added papiServiceGetAttributeList() call to retrieve print service and implementation
- 3547 specific information from a service handle.
- 3548 Added a “Conformance” section to the document. A draft introduction and conformance
- 3549 table are included, but the actual conformance levels need work. The bulk of this was
- 3550 included from Ira's and Tom's draft.
- 3551 Moved Attribute section in front of the Service, Printer, and Job sections interfaces to
- 3552 improve flow of document.
- 3553 Added papi_encryption_t to common structures
- 3554 Added constraints chapter. The bulk of this chapter was copied directly from v0.3 of the
- 3555 papi capabilities document.

Chapter 15: Change History

3556 **15.5 Version 0.9 (November 18, 2002).**

- 3557 Changed media-margins order to "top, right, bottom, left" to match other standards.
- 3558 Changed media-margins units to "100ths of millimeters" to match other standards. Also,
- 3559 reworded last paragraph of description of this attribute.

3560 **15.6 Version 0.8 (November 15, 2002).**

- 3561 Added value field, explanation, and corrected example for `papi_filter_t`.
- 3562 Added media-margins attribute to "Extension Attributes" section.
- 3563 Renamed function names with "Username" to "UserName", and renamed function names
- 3564 with "Servicename" to "ServiceName", and Miscellaneous wording and typo corrections.

3565 **15.7 Version 0.7 (October 18, 2002).**

- 3566 Added `attr_delim` argument to `papiAttributeListToString` and made new-line ("`\n`") an
- 3567 allowed attribute delimiter on input to `papiAttributeListFromString`.
- 3568 Added "Semantics Reference" subsections to functions.
- 3569 Added to References: [5101.1], [RFC3196], and URIs for obtaining IPP documents.
- 3570 Added `PAPI_JOB_TICKET_NOT_SUPPORTED` status code.
- 3571 Added "Globalization" section in the "Print System Model" chapter.
- 3572 Changed definition and usage of returned value from `papiAttributeListGetValue`. Also
- 3573 clarified what happens to output values when a `papiAttributeListGet*` call has an error.
- 3574 Clarified descriptions of `papiPrinterGetAttributeList` and `papiJobGetAttributeList`.
- 3575 Changed buffer length arguments from `int` to `size_t`.
- 3576 Clarified that `papiServiceDestroy` must always be called after a call to `papiServiceCreate`.
- 3577 Removed `attributes-charset`, `attributes-natural-language`, and `job-printer-up-time` from the
- 3578 "Required Job Attributes" (they may be hidden inside the PAPI implementation).
- 3579 Clarified result of passing both attributes and a job ticket on all the job submission
- 3580 functions.
- 3581 Miscellaneous wording and typo corrections.

3582 **15.8 Version 0.6 (September 20, 2002)**

- 3583 Made explanation of `requested_attrs` in `papiPrintersList` the same as it is for
- 3584 `papiPrinterQuery`.
- 3585 Moved `units` argument on `papiAttributeListAddResolution` to the end of the argument list to
- 3586 match the corresponding get function.

3587 Added papiAttributeListAddCollection and papiAttributeListGetCollection.

3588 Removed unneeded extra level of indirection from attrs argument to papiAttributeListGet*
3589 functions. Also made the attrs argument const.

3590 Added note to "Conventions" section that strings are assumed to be UTF-8 encoded.

3591 Added papiAttributeListFromString and papiAttributeListToString functions, along with a
3592 new appendix defining the string format syntax.

3593 Added papiJobSubmitByReference, papiJobStreamOpen, papiJobStreamWrite, and
3594 papiJobStreamClose functions.

3595 Added short "Document" section in the "Print System Model" chapter.

3596 Added explanation of how multiple files specified in the papiJobSubmit file_names
3597 argument are handled by the print system.

3598 Changed papi_job_ticket_t "uri" field to "file_name" and added explanation text.

3599 Added explanation of implementation option for merging papiJobSubmit attributes with
3600 job_ticket argument.

3601 Added "References" appendix.

3602 Added "IPP Attribute Type Mapping" appendix.

3603 Added "PWG" job ticket format to papi_jt_format_t.

3604 Miscellaneous wording and typo corrections.

3605 **15.9 Version 0.5 (August 30, 2002).**

3606 Added job_attrs argument to papiPrinterQuery to support more accurate query of printer
3607 capabilities.

3608 Added management functions papiAttributeDelete, papiJobModify, and papiPrinterModify.

3609 Added functions papiAttributeListGetValue, papiAttributeListGetString,
3610 papiAttributeListGetInteger, etc.

3611 Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent with the
3612 naming convention (first word after "papi" is the object being operated upon).

3613 Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.

3614 Made description of authentication more implementation-independent.

3615 Added reference to IPP attributes summary document.

3616 Added result argument to papiPrinterPurgeJobs.

3617 Added "collection attribute" support (PAPI_COLLECTION type).

3618 Changed boolean values to consistently use char. Added PAPI_FALSE and PAPI_TRUE
3619 enum values.

Chapter 15: Change History

3620 **15.10 Version 0.4 (July 19, 2002).**

3621 Made papi_job_t and papi_printer_t opaque handles and added "get" functions to access the
3622 associated information (papiPrinterGetAttributeList, papiJobGetAttributeList,
3623 papiJobGetId, papiJobGetPrinterName, papiJobGetJobTicket).

3624 Removed variable length argument lists from attribute add functions.

3625 Changed order and name of flag value passed to attribute add functions.

3626 Eliminated indirection in date/time value passed to papiAttributeAddDatetime.

3627 Added message argument to papiPrinterPause.

3628 **15.11 Version 0.3 (June 24, 2002).**

3629 Converted to DocBook format from Microsoft Word

3630 Major rewrite, including:

3631 Changed how printer names are described in "Model/Printer"

3632 Changed fixed length strings to pointers in numerous structures/sections

3633 Redefined attribute/value structures and associated API descriptions

3634 Changed list/query functions to return "objects"

3635 Rewrote "Attributes API" chapter

3636 Changed many function definitions to pass NULL-terminated arrays of pointers instead of a
3637 separate count argument

3638 Changed papiJobSubmit to take an attribute list structure as input instead of a formatted
3639 string

3640 **15.12 Version 0.2 (April 17, 2002).**

3641 Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)

3642 Filled in "Encryption" section and added information about encryption in "Object
3643 Identification" section

3644 Added "short_name" field in "Object Identification" section

3645 Added "Job Ticket (papi_job_ticket_t)" section

3646 Added papiPrinterPause

3647 Added papiPrinterResume

3648 Added papiPurgeJobs

3649 Added optional job_ticket argument to papiJobSubmit

3650 Added optional passing of filenames by URI to papiJobSubmit

- 3651 Added papiHoldJob
- 3652 Added papiReleaseJob
- 3653 Added papiRestartJob
- 3654 **15.13 Version 0.1 (April 3, 2002).**
- 3655 Original draft version