

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.....	6
Chapter 2: Print System Model.....	7
2.1 Introduction.....	7
2.2 Model.....	7
2.3 Security.....	8
2.4 Globalization.....	9
Chapter 3: Common Structures.....	10
3.1 Conventions.....	10
3.2 Service Object (papi_service_t).....	10
3.3 Attributes and Values (papi_attribute_t).....	10
3.4 Job Object (papi_job_t).....	12
3.5 Stream Object (papi_stream_t).....	12
3.6 Printer Object (papi_printer_t).....	12
3.7 Job Ticket (papi_job_ticket_t).....	12
3.8 Status (papi_status_t).....	13
3.9 List Filter (papi_filter_t).....	14
3.10 Encryption (papi_encrypt_t).....	16
Chapter 4: Attributes API.....	17
4.1 papiAttributeListAdd.....	18
4.2 papiAttributeListAddString.....	19
4.3 papiAttributeListAddInteger.....	20
4.4 papiAttributeListAddBoolean.....	22
4.5 papiAttributeListAddRange.....	23
4.6 papiAttributeListAddResolution.....	24
4.7 papiAttributeListAddDatetime.....	26
4.8 papiAttributeListAddCollection.....	27
4.9 papiAttributeListAddMetadata.....	28
4.10 papiAttributeListDelete.....	30
4.11 papiAttributeListGetValue.....	31
4.12 papiAttributeListGetString.....	32
4.13 papiAttributeListGetInteger.....	34
4.14 papiAttributeListGetBoolean.....	35
4.15 papiAttributeListGetRange.....	36
4.16 papiAttributeListGetResolution.....	38
4.17 papiAttributeListGetDatetime.....	39
4.18 papiAttributeListGetCollection.....	41
4.19 papiAttributeListGetMetadata.....	42
4.20 papiAttributeListFree.....	43
4.21 papiAttributeListFind.....	44
4.22 papiAttributeListGetNext.....	45
4.23 papiAttributeListFromString.....	46
4.24 papiAttributeListToString.....	47

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 0.91 (January 28, 2004).....	129
15.2 Version 0.9 (November 18, 2002).....	129
15.3 Version 0.8 (November 15, 2002).....	129
15.4 Version 0.7 (October 18, 2002).....	129
15.5 Version 0.6 (September 20, 2002).....	130
15.6 Version 0.5 (August 30, 2002).....	131
15.7 Version 0.4 (July 19, 2002).....	131
15.8 Version 0.3 (June 24, 2002).....	131
15.9 Version 0.2 (April 17, 2002).....	132
15.10 Version 0.1 (April 3, 2002).....	132

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](#).

38 Chapter 2: Print System Model

39 2.1 Introduction

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

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

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

49 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	

50

51 2.2 Model

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

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

59 2.2.1 Print Service

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

63 2.2.2 Printer

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

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

Chapter 2: Print System Model

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

72 **2.2.3 Job**

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

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

79 **2.2.4 Document**

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

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

87 **2.3 Security**

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

90 **2.3.1 Authentication**

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

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

97 **2.3.2 Authorization**

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

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

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

105 **2.3.3 Encryption**

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

111 **2.4 Globalization**

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

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

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

122 Chapter 3: Common Structures

123 3.1 Conventions

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

129 3.2 Service Object (*papi_service_t*)

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

```
133 typedef void *papi_service_t;
```

134

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

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

143

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

146 3.3 Attributes and Values (*papi_attribute_t*)

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

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

```

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

```

Chapter 3: Common Structures

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

208

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

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

215

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

217 **3.4 Job Object (papi_job_t)**

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

222 **3.5 Stream Object (papi_stream_t)**

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

225 **3.6 Printer Object (papi_printer_t)**

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

230 **3.7 Job Ticket (papi_job_ticket_t)**

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

```

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

```

249

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

257 **3.8 Status (*papi_status_t*)**

```

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

```

Chapter 3: Common Structures

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

305

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

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

310 **3.9 List Filter (*papi_filter_t*)**

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

```
314 typedef enum {
```

```

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

```

330

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

```

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

```

341 The filter bitmask values are:

```

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

```

Chapter 3: Common Structures

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

364 **3.10 Encryption (*papi_encrypt_t*)**

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

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

373

374 Chapter 4: Attributes API

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

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

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

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

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

414 **4.1 *papiAttributeListAdd***

415 **4.1.1 Description**

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

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

425 **4.1.2 Syntax**

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

429 **4.1.3 Inputs**

430 **4.1.3.1 *attrs***

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

432 **4.1.3.2 *add_flags***

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

435 **4.1.3.3 *name***

436 Points to the name of the attribute to add.

437 **4.1.3.4 *type***

438 The type of values for this attribute.

439 **4.1.3.5 *value***

440 Points to the attribute value to be added.

441 **4.1.4 Outputs**

442 **4.1.4.1 *attrs***

443 The attribute list is updated.

444 **4.1.5 Returns**

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

447 **4.1.6 Example**

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

457 **4.1.7 See Also**

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

462 **4.2 *papiAttributeListAddString***

463 **4.2.1 Description**

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

468 **4.2.2 Syntax**

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

471 **4.2.3 Inputs**

472 **4.2.3.1 *attrs***

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

474 **4.2.3.2 *add_flags***

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

477 **4.2.3.3 *name***

478 Points to the name of the attribute to add.

479 **4.2.3.4 *value***

480 The string value to be added to the attribute.

481 **4.2.4 Outputs**

482 **4.2.4.1 *attrs***

483 The attribute list is updated.

484 **4.2.5 Returns**

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

487 **4.2.6 Example**

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

495 **4.2.7 See Also**

496 [papiAttributeListAdd](#), [papiAttributeListFree](#)

497 **4.3 *papiAttributeListAddInteger***

498 **4.3.1 Description**

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

503 **4.3.2 Syntax**

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

506 **4.3.3 Inputs**

507 **4.3.3.1 attrs**

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

509 **4.3.3.2 add_flags**

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

512 **4.3.3.3 name**

513 Points to the name of the attribute to add.

514 **4.3.3.4 value**

515 The integer value to be added to the attribute.

516 **4.3.4 Outputs**

517 **4.3.4.1 attrs**

518 The attribute list is updated.

519 **4.3.5 Returns**

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

522 **4.3.6 Example**

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

530 **4.3.7 See Also**

531 [papiAttributeListAdd](#), [papiAttributeListFree](#)

532 **4.4 *papiAttributeListAddBoolean***

533 **4.4.1 Description**

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

538 **4.4.2 Syntax**

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

541 **4.4.3 Inputs**

542 **4.4.3.1 *attrs***

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

544 **4.4.3.2 *add_flags***

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

547 **4.4.3.3 *name***

548 Points to the name of the attribute to add.

549 **4.4.3.4 *value***

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

551 **4.4.4 Outputs**

552 **4.4.4.1 *attrs***

553 The attribute list is updated.

554 **4.4.5 Returns**

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

557 **4.4.6 Example**

```
558 papi_status_t status;  
559 papi_attribute_t **attrs = NULL;
```

```
560 ...
561 status = papiAttributeListAddBoolean(&attrs, PAPI_ATTR_EXCL,
562                                     "color-supported", PAPI_TRUE);
563 ...
564 papiAttributeListFree(attrs);
```

565 **4.4.7 See Also**

566 [papiAttributeListAdd](#), [papiAttributeListFree](#)

567 **4.5 *papiAttributeListAddRange***

568 **4.5.1 Description**

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

573 **4.5.2 Syntax**

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

576 **4.5.3 Inputs**

577 **4.5.3.1 *attrs***

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

579 **4.5.3.2 *add_flags***

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

582 **4.5.3.3 *name***

583 Points to the name of the attribute to add.

584 **4.5.3.4 *lower***

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

587 **4.5.3.5 *upper***

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

590 **4.5.4 Outputs**

591 **4.5.4.1 *attrs***

592 The attribute list is updated.

593 **4.5.5 Returns**

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

596 **4.5.6 Example**

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

604 **4.5.7 See Also**

605 [papiAttributeListAdd](#), [papiAttributeListFree](#)

606 **4.6 *papiAttributeListAddResolution***

607 **4.6.1 Description**

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

612 **4.6.2 Syntax**

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

616 **4.6.3 Inputs**

617 **4.6.3.1 *attrs***

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

619 **4.6.3.2 add_flags**

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

622 **4.6.3.3 name**

623 Points to the name of the attribute to add.

624 **4.6.3.4 xres**

625 The integer X-axis resolution value.

626 **4.6.3.5 yres**

627 The integer Y-axis resolution value.

628 **4.6.3.6 Units**

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

630 **4.6.4 Outputs**

631 **4.6.4.1 attrs**

632 The attribute list is updated.

633 **4.6.5 Returns**

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

636 **4.6.6 Example**

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

645 **4.6.7 See Also**

646 [papiAttributeListAdd](#), [papiAttributeListFree](#)

647 **4.7 *papiAttributeListAddDatetime***

648 **4.7.1 Description**

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

653 **4.7.2 Syntax**

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

656 **4.7.3 Inputs**

657 **4.7.3.1 *attrs***

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

659 **4.7.3.2 *add_flags***

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

662 **4.7.3.3 *name***

663 Points to the name of the attribute to add.

664 **4.7.3.4 *value***

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

666 **4.7.4 Outputs**

667 **4.7.4.1 *attrs***

668 The attribute list is updated.

669 **4.7.5 Returns**

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

672 **4.7.6 Example**

```
673 papi_status_t status;  
674 papi_attribute_t **attrs = NULL;
```

```

675 time_t date_time;
676 ...
677 time(&date_time);
678 status = papiAttributeListAdd(&attrs, PAPI_EXCL,
679                               "date-time-at-creation", date_time);
680 ...
681 papiAttributeListFree(attrs);

```

682 **4.7.7 See Also**

683 [papiAttributeListAdd](#), [papiAttributeListFree](#)

684 **4.8 papiAttributeListAddCollection**

685 **4.8.1 Description**

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

691 **4.8.2 Syntax**

```

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

```

695 **4.8.3 Inputs**

696 **4.8.3.1 *attrs***

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

698 **4.8.3.2 *add_flags***

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

701 **4.8.3.3 *name***

702 Points to the name of the attribute to add.

703 **4.8.3.4 *collection***

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

705 **4.8.4 Outputs**

706 **4.8.4.1 attrs**

707 The attribute list is updated.

708 **4.8.5 Returns**

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

711 **4.8.6 Example**

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

728 **4.8.7 See Also**

729 [papiAttributeListAdd](#), [papiAttributeListFree](#)

730 **4.9 papiAttributeListAddMetadata**

731 **4.9.1 Description**

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

736 **4.9.2 Syntax**

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

740 **4.9.3 Inputs**

741 **4.9.3.1 *attrs***

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

743 **4.9.3.2 *add_flags***

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

746 **4.9.3.3 *name***

747 Points to the name of the attribute to add.

748 **4.9.3.4 *value***

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

754 **4.9.4 Outputs**

755 **4.9.4.1 *attrs***

756 The attribute list is updated.

757 **4.9.5 Returns**

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

760 **4.9.6 Example**

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

770 **4.9.7 See Also**

771 [papiAttributeListAdd](#), [papiAttributeListFree](#)

772 **4.10 *papiAttributeListDelete***

773 **4.10.1 Description**

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

776 **4.10.2 Syntax**

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

778 **4.10.3 Inputs**

779 **4.10.3.1 *attrs***

780 Points to an attribute list.

781 **4.10.3.2 *name***

782 Points to the name of the attribute to remove.

783 **4.10.4 Outputs**

784 **4.10.4.1 *attrs***

785 The attribute list is updated.

786 **4.10.5 Returns**

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

789 **4.10.6 Example**

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

796 **4.10.7 See Also**

797 [papiAttributeListFree](#)

798 **4.11 *papiAttributeListGetValue***

799 **4.11.1 Description**

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

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

805 **4.11.2 Syntax**

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

809 **4.11.3 Inputs**

810 **4.11.3.1 *attrs***

811 Points to an attribute list.

812 **4.11.3.2 *iterator***

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

817 **4.11.3.3 *name***

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

820 **4.11.3.4 *type***

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

823 **4.11.4 Outputs**

824 **4.11.4.1 *iterator***

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

826 **4.11.4.2 *value***

827 Points to the variable where a pointer to the attribute value is to be returned. Note that the

Chapter 4: Attributes API

828 returned pointer points to the attribute's value in the list (no copy of the value is made) so
829 that the caller does not need to do any special cleanup of the returned value's memory (it is
830 cleaned up when the containing attribute list is deallocated).
831 If this call returns an error, the output value is not changed.

832 **4.11.5 Returns**

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

835 **4.11.6 Example**

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

844 **4.11.7 See Also**

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

848 **4.12 *papiAttributeListGetString***

849 **4.12.1 Description**

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

851 **4.12.2 Syntax**

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

854 **4.12.3 Inputs**

855 **4.12.3.1 *attrs***

856 Points to an attribute list.

857 **4.12.3.2 *iterator***

858 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only

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

862 **4.12.3.3 name**

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

867 **4.12.4 Outputs**

868 **4.12.4.1 Iterator**

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

870 **4.12.4.2 value**

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

876 **4.12.5 Returns**

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

879 **4.12.6 Example**

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

888 **4.12.7 See Also**

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

890 **4.13 *papiAttributeListGetInteger***

891 **4.13.1 Description**

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

893 **4.13.2 Syntax**

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

896 **4.13.3 Inputs**

897 **4.13.3.1 *attrs***

898 Points to an attribute list.

899 **4.13.3.2 *iterator***

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

904 **4.13.3.3 *name***

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

909 **4.13.4 Outputs**

910 **4.13.4.1 *iterator***

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

912 **4.13.4.2 *value***

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

915 **4.13.5 Returns**

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

918 **4.13.6 Example**

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

926 **4.13.7 See Also**

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

928 **4.14 papiAttributeListGetBoolean**

929 **4.14.1 Description**

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

931 **4.14.2 Syntax**

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

934 **4.14.3 Inputs**

935 **4.14.3.1 *attrs***

936 Points to an attribute list.

937 **4.14.3.2 *iterator***

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

942 **4.14.3.3 *name***

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

947 **4.14.4 Outputs**

948 **4.14.4.1 Iterator**

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

950 **4.14.4.2 value**

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

953 **4.14.5 Returns**

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

956 **4.14.6 Example**

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

965 **4.14.7 See Also**

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

967 **4.15 papiAttributeListGetRange**

968 **4.15.1 Description**

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

970 **4.15.2 Syntax**

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

973 **4.15.3 Inputs**

974 **4.15.3.1 attrs**

975 Points to an attribute list.

976 **4.15.3.2 iterator**

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

981 **4.15.3.3 name**

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

986 **4.15.4 Outputs**

987 **4.15.4.1 iterator**

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

989 **4.15.4.2 lower**

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

992 **4.15.4.3 upper**

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

995 **4.15.5 Returns**

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

998 **4.15.6 Example**

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

1008 **4.15.7 See Also**

1009 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1010 **4.16 papiAttributeListGetResolution**

1011 **4.16.1 Description**

1012 Get a resolution-valued attribute's value from an attribute list.

1013 **4.16.2 Syntax**

```
1014 papi_status_t papiAttributeListGetResolution(papi_attribute_t **attrs, void **iterator,  
1015                                             char *name, int *xres, int *yres, papi_res_t *units);
```

1016 **4.16.3 Inputs**

1017 **4.16.3.1 attrs**

1018 Points to an attribute list.

1019 **4.16.3.2 iterator**

1020 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1021 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1022 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1023 be passed in unchanged on subsequent calls to this function to get the remaining values.

1024 **4.16.3.3 name**

1025 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1026 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1027 found in the attribute list, but is not a PAPI_RESOLUTION, PAPI_NOT_POSSIBLE will
1028 be returned.

1029 **4.16.4 Outputs**

1030 **4.16.4.1 iterator**

1031 See [iterator](#) in the [Inputs](#) section above.

1032 **4.16.4.2 xres**

1033 Pointer to the int where the X-resolution value is returned. The value from the attribute list
1034 is copied to this location. If this call returns an error, the output value is not changed.

1035 **4.16.4.3 yres**

1036 Pointer to the int where the Y-resolution value is returned. The value from the attribute list
1037 is copied to this location. If this call returns an error, the output value is not changed.

1038 **4.16.4.4 units**

1039 Pointer to the variable where the resolution-units value is returned. The value from the
1040 attribute list is copied to this location. If this call returns an error, the output value is not
1041 changed.

1042 **4.16.5 Returns**

1043 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1044 returned.

1045 **4.16.6 Example**

```
1046 papi_status_t status;  
1047 papi_attribute_t **attrs = NULL;  
1048 int xres, yres;  
1049 papi_res_t units;  
1050 ...  
1051 status = papiAttributeListGetResolution(attrs, NULL,  
1052                                     "printer-resolution", &xres, &yres, &units);  
1053 ...  
1054 papiAttributeListFree(attrs);
```

1055 **4.16.7 See Also**

1056 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1057 **4.17 papiAttributeListGetDatetime**

1058 **4.17.1 Description**

1059 Get a datetime-valued attribute's value from an attribute list.

1060 **4.17.2 Syntax**

```
1061 papi_status_t papiAttributeListGetDatetime(papi_attribute_t **attrs, void **iterator,  
1062                                           char *name, time_t *value);
```

1063 **4.17.3 Inputs**

1064 **4.17.3.1 attrs**

1065 Points to an attribute list.

1066 **4.17.3.2 iterator**

1067 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1068 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1069 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1070 be passed in unchanged on subsequent calls to this function to get the remaining values.

1071 **4.17.3.3 name**

1072 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1073 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1074 found in the attribute list, but is not a PAPI_DATETIME, PAPI_NOT_POSSIBLE will be
1075 returned.

1076 **4.17.4 Outputs**

1077 **4.17.4.1 iterator**

1078 See [iterator](#) in the [Inputs](#) section above

1079 **4.17.4.2 value**

1080 Pointer to the time_t where the value is returned. The value from the attribute list is copied
1081 to this location. If this call returns an error, the output value is not changed.

1082 **4.17.5 Returns**

1083 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1084 returned.

1085 **4.17.6 Example**

```
1086 papi_status_t status;  
1087 papi_attribute_t **attrs = NULL;  
1088 time_t value = 0;  
1089 ...  
1090 status = papiAttributeListGetDatetime(attrs, NULL,  
1091                                     "date-time-at-creation", &value);  
1092 ...  
1093 papiAttributeListFree(attrs);
```

1094 **4.17.7 See Also**

1095 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1096 **4.18 papiAttributeListGetCollection**

1097 **4.18.1 Description**

1098 Get a collection-valued attribute's value from an attribute list.

1099 **4.18.2 Syntax**

```
1100 papi_status_t papiAttributeListGetCollection(papi_attribute_t **attrs, void **iterator,  
1101 char *name, papi_attribute_t ***value);
```

1102 **4.18.3 Inputs**

1103 **4.18.3.1 attrs**

1104 Points to an attribute list.

1105 **4.18.3.2 iterator**

1106 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1107 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1108 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1109 be passed in unchanged on subsequent calls to this function to get the remaining values.

1110 **4.18.3.3 name**

1111 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1112 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is
1113 found in the attribute list, but is not a PAPI_COLLECTION, PAPI_NOT_POSSIBLE will
1114 be returned.

1115 **4.18.3.4 type**

1116 The type of values for this attribute.

1117 **4.18.4 Outputs**

1118 **4.18.4.1 Iterator**

1119 See [iterator](#) in the [Inputs](#) section above.

1120 **4.18.4.2 value**

1121 Points to the variable where a pointer to the attribute value is to be returned. Note that the
1122 returned pointer points to the attribute's value in the list (no copy of the value is made) so
1123 that the caller does not need to do any special cleanup of the returned value's memory (it is
1124 cleaned up when the containing attribute list is deallocated).
1125 If this call returns an error, the output value is not changed.

1126 **4.18.5 Returns**

1127 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1128 returned.

1129 **4.18.6 Example**

```
1130 papi_status_t status;  
1131 papi_attribute_t **attrs = NULL;  
1132 papi_attribute_t **value = NULL;  
1133 ...  
1134 status = papiAttributeListGetCollection(attrs, NULL,  
1135                                     "media-col", &value);  
1136 ...  
1137 papiAttributeListFree(attrs);
```

1138 **4.18.7 See Also**

1139 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1140 **4.19 papiAttributeListGetMetadata**

1141 **4.19.1 Description**

1142 Get a meta-valued attribute's value from an attribute list.

1143 **4.19.2 Syntax**

```
1144 papi_status_t papiAttributeListGetMetadata(papi_attribute_t **attrs, void **iterator,  
1145                                           char *name, papi_metadata_t *value);
```

1146 **4.19.3 Inputs**

1147 **4.19.3.1 attrs**

1148 Points to an attribute list.

1149 **4.19.3.2 iterator**

1150 (optional) Pointer to an opaque (void*) value iterator. If the argument is NULL then only
1151 the first value is returned, even if the attribute is multi-valued. If the argument points to a
1152 void* that is set to NULL, then the first attribute value is returned and the iterator can then
1153 be passed in unchanged on subsequent calls to this function to get the remaining values.

1154 **4.19.3.3 name**

1155 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1156 supplied attribute list, PAPI_NOT_FOUND will be returned. If the named attribute is

1157 found in the attribute list, but is not a PAPI_STRING, PAPI_NOT_POSSIBLE will be
1158 returned.

1159 **4.19.3.4 type**

1160 The type of values for this attribute.

1161 **4.19.4 Outputs**

1162 **4.19.4.1 Iterator**

1163 See [iterator](#) in the [Inputs](#) section above.

1164 **4.19.4.2 value**

1165 Points to the variable where the attribute value is to be returned. If this call returns an error,
1166 the output value is not changed.

1167 **4.19.5 Returns**

1168 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1169 returned.

1170 **4.19.6 Example**

```
1171 papi_status_t status;  
1172 papi_attribute_t **attrs = NULL;  
1173 papi_metadata_t value = PAPI_NO_VALUE;  
1174 ...  
1175 status = papiAttributeListGetMetadata(attrs, NULL,  
1176                                     "media", &value);  
1177 ...  
1178 papiAttributeListFree(attrs);
```

1179 **4.19.7 See Also**

1180 [papiAttributeListGetValue](#), [papiAttributeListFree](#)

1181 **4.20 papiAttributeListFree**

1182 **4.20.1 Description**

1183 Frees an attribute list

1184 **4.20.2 Syntax**

```
1185 void papiAttributeListFree(papi_attribute_t **attrs);
```

1186 **4.20.3 Inputs**

1187 **4.20.3.1 *attrs***

1188 Attribute list to be deallocated.

1189 **4.20.4 Outputs**

1190 none

1191 **4.20.5 Returns**

1192 none

1193 **4.20.6 Example**

```
1194 papi_attribute_t **attrs = NULL;  
1195 ...  
1196 papiAttributeListFree(attrs);
```

1197 **4.20.7 See Also**

1198 [papiAttributeListAdd](#), [papiAttributeListAddString](#), [papiAttributeListAddInteger](#),
1199 [papiAttributeListAddBoolean](#), [papiAttributeListAddRange](#)
1200 [papiAttributeListAddResolution](#), [papiAttributeListAddDatetime](#),
1201 [papiAttributeListAddCollection](#), [papiAttributeListFromString](#), [papiAttributeListFree](#)

1202 **4.21 *papiAttributeListFind***

1203 **4.21.1 Description**

1204 Find an attribute in an attribute list.

1205 **4.21.2 Syntax**

```
1206 papi_attribute_t *papiAttributeListFind(papi_attribute_t **attrs, char *name);
```

1207 **4.21.3 Inputs**

1208 **4.21.3.1 *attrs***

1209 Points to an attribute list.

1210 **4.21.3.2 *name***

1211 Points to the name of the attribute to retrieve. If the named attribute can not be found in the
1212 supplied attribute list, PAPI_NOT_FOUND will be returned.

1213 **4.21.4 Outputs**

1214 none

1215 **4.21.5 Returns**

1216 Pointer to the named attribute found in the attribute list. The result will be deallocated
1217 when the containing attribute list is destroyed. NULL indicates that the specified attribute
1218 was not found

1219 **4.21.6 Example**

```
1220 papi_attribute_t **attrs = NULL;  
1221 papi_attribute_t *value;  
1222 ...  
1223 value = papiAttributeListFind(attrs, "job-name");  
1224 ...  
1225 papiAttributeListFree(attrs);
```

1226 **4.21.7 See Also**

1227 [papiAttributeListGetValue](#)

1228 **4.22 *papiAttributeListGetNext***

1229 **4.22.1 Description**

1230 Get the next attribute in an attribute list.

1231 **4.22.2 Syntax**

```
1232 papi_attribute_t **papiAttributeListGetNext(papi_attribute_t **attrs, void **iterator);
```

1233 **4.22.3 Inputs**

1234 **4.22.3.1 *attrs***

1235 Points to an attribute list.

1236 **4.22.3.2 *iterator***

1237 Pointer to an opaque (void*) iterator. This should be NULL to find the first attribute and
1238 then passed in unchanged on subsequent calls to this function.

1239 **4.22.4 Outputs**

1240 **4.22.4.1 Iterator**

1241 See [iterator](#) in the [Inputs](#) section above.

1242 **4.22.5 Returns**

1243 Pointer to the next attribute in the attribute list. The result will be deallocated when the
1244 containing attribute list is destroyed. NULL indicates that the end of the attribute list was
1245 reached

1246 **4.22.6 Example**

```
1247 papi_attribute_t **attrs = NULL;  
1248 papi_attribute_t *value;  
1249 void *iterator = NULL;  
1250 ...  
1251 while ((value = papiAttributeGetNext(attrs, &iterator)) != NULL) {  
1252     ...  
1253 }  
1254 ...  
1255 papiAttributeListFree(attrs);
```

1256 **4.22.7 See Also**

1257 [papiAttributeListFind](#)

1258 **4.23 papiAttributeListFromString**

1259 **4.23.1 Description**

1260 Convert a string of text options to an attribute list. PAPI provides two functions which map
1261 job attributes to and from text options that are typically provided on the command-line by
1262 the user. This text encoding is also backwards-compatible with existing printing systems
1263 and is relatively simple to parse and generate. See [Attribute List Text Representation](#) for a
1264 definition of the string syntax.

1265 **4.23.2 Syntax**

```
1266 papi_status_t papiAttributeListFromString(papi_attribute_t*** attrs,  
1267                                           int add_flags, char* buffer);
```

1268 **4.23.3 Inputs**

1269 **4.23.3.1 *attrs***

1270 Points to an attribute list. If **attrs* is NULL then this function will allocate the attribute list.

1271 **4.23.3.2 *add_flags***

1272 A mask field consisting of one or more PAPI_ATTR_* values OR-ed together that
1273 indicates how to handle the request.

1274 **4.23.3.3 *buffer***

1275 Points to text options.

1276 **4.23.4 Outputs**

1277 **4.23.4.1 *attrs***

1278 The attribute list is updated.

1279 **4.23.5 Returns**

1280 If the text string is successfully converted to an attribute list, a value of PAPI_OK is
1281 returned. Otherwise an appropriate failure value is returned.

1282 **4.23.6 Example**

```
1283 papi_status_t status;  
1284 papi_attribute_t **attrs = NULL;  
1285 char *string = "copies=1 job-name=John\'s\ Really\040Nice\ Job";  
1286 ...  
1287 status = papiAttributeListFromString(attrs, PAPI_ATTR_EXCL, string);  
1288 ...  
1289 papiAttributeListFree(attrs);
```

1290 **4.23.7 See Also**

1291 [papiAttributeListFind](#)

1292 **4.24 *papiAttributeListToString***

1293 **4.24.1 Description**

1294 Convert an attribute list to its text representation. The destination string is limited to at most
1295 (buflen - 1) bytes plus the trailing null byte.

1296 PAPI provides two functions which map job attributes to and from text options that are
1297 typically provided on the command-line by the user. This text encoding is also backwards-

1298 compatible with existing printing systems and is relatively simple to parse and generate.
1299 See [Attribute List Text Representation](#) for a definition of the string syntax.

1300 **4.24.2 Syntax**

```
1301 papi_status_t papiAttributeListToString(papi_attribute_t** attrs,  
1302                                       char* attr_delim, char* buffer,  
1303                                       size_t buflen );
```

1304 **4.24.3 Inputs**

1305 **4.24.3.1 attr**

1306 Points to an attribute list.

1307 **4.24.3.2 attr_delim**

1308 (optional) If not NULL, points to a string to be placed between attributes in the output
1309 buffer. If NULL, a space is used as the attribute delimiter.

1310 **4.24.3.3 buffer**

1311 Points to a string buffer to receive the to receive the text representation of the attribute list.

1312 **4.24.3.4 buflen**

1313 Specifies the length of the string buffer in bytes.

1314 **4.24.4 Outputs**

1315 **4.24.4.1 buffer**

1316 The buffer is filled with the text representation of the attribute list. The buffer will always
1317 be set to something by this function (buffer[0] = NULL in cases of an error).

1318 **4.24.5 Returns**

1319 If the attribute list is successfully converted to a text string, a value of PAPI_OK is
1320 returned. Otherwise an appropriate failure value is returned.

1321 **4.24.6 Example**

```
1322 papi_attribute_t **attrs = NULL;  
1323 char buffer[8192];  
1324 ...  
1325 papiAttributeListToString(attrs, NULL, buffer, sizeof (buffer));  
1326 ...  
1327 papiAttributeListFree(attrs);
```


1328 **4.24.7 See Also**

1329 [PapiAttributeListFromString](#)

1330 **Chapter 5: Service API**

1331 The service segment of the PAPI provides a means of creating, modifying, or destroying a
1332 context (or object) used to interact with a print service. This context is opaque to
1333 applications using it and may be used by implementations to store internal data such as file
1334 or socket descriptors, operation results, credentials, etc.

1335 **5.1 *papiServiceCreate***

1336 **5.1.1 Description**

1337 Create a print service handle to be used in subsequent calls. Memory is allocated and
1338 copies of the input arguments are created so that the handle can be used outside the scope of
1339 the input variables.
1340 The caller must call [papiServiceDestroy](#) when done in order to free the resources associated
1341 with the print service handle. This must be done even if the `papiServiceCreate` call failed,
1342 because a service creation failure may have resulted in a partial service context with
1343 additional error information.

1344 **5.1.2 Syntax**

```
1345 papi_status_t papiServiceCreate( papi_service_t *handle, char *service_name,  
1346                               char *user_name, char *password,  
1347                               int (*authCB)(papi_service_t svc),  
1348                               papi_encryption_t encryption, void *app_data );
```

1349 **5.1.3 Inputs**

1350 **5.1.3.1 *service_name***

1351 (optional) Points to the name or URI of the service to use. A NULL value indicates that a
1352 “default service” should be used (the configuration of a default service is implementation-
1353 specific and may consist of environment variables, config files, etc. Default service
1354 selection is not addressed by this standard).

1355 **5.1.3.2 *user_name***

1356 (optional) Points to the name of the user who is making the requests. A NULL value
1357 indicates that the user name associated with the process in which the API call is made
1358 should be used.

1359 **5.1.3.3 *Password***

1360 (optional) Points to the password to be used to authenticate the user to the print service.

1361 **5.1.3.4 AuthCB**

1362 (optional) Points to a callback function to be used in authenticating the user to the print
1363 service if no password was supplied (or user input is required). A NULL value indicates
1364 that no callback should be made. The callback function should return 0 if the request is to
1365 be canceled and non-zero if new authentication information has been set.

1366 **5.1.3.5 Encryption**

1367 Specifies the encryption type to be used by the PAPI functions.

1368 **5.1.3.6 app_data**

1369 (optional) Points to application-specific data for use by the callback. The caller is
1370 responsible for allocating and freeing memory associated with this data.

1371 **5.1.4 Outputs**

1372 **5.1.4.1 handle**

1373 A print service handle to be used on subsequent API calls. The handle will always be set to
1374 something even if the function fails. In the event that the function fails, the handle may be
1375 set to NULL or it may be set to a valid handle that contains error information.

1376 **5.1.5 Returns**

1377 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1378 returned.

1379 **5.1.6 Example**

```
1380 papi_status_t status;  
1381 papi_service_t handle = NULL;  
1382 ...  
1383 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1384                             "user", "password", NULL,  
1385                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1386 ...  
1387 papiServiceDestroy(handle);
```

1388 **5.1.7 See Also**

1389 [papiServiceDestroy](#), [papiServiceGetStatusMessage](#), [papiServiceSetUserName](#),
1390 [papiServiceSetPassword](#), [papiServiceSetEncryption](#), [papiServiceSetAuthCB](#),
1391 [papiServiceSetAppData](#), [papiServiceGetStatusMessage](#)

1392 **5.2 *papiServiceDestroy***

1393 **5.2.1 Description**

1394 Destroy a print service handle and free the resources associated with it. This must be called
1395 even if the [papiServiceCreate](#) call failed, because there may be error information associated
1396 with the returned handle. If there is application data associated with the service handle, it is
1397 the caller's responsibility to free this memory.

1398 **5.2.2 Syntax**

```
1399 void papiServiceDestroy(papi_service_t handle);
```

1400 **5.2.3 Inputs**

1401 **5.2.3.1 *handle***

1402 The print service handle to be destroyed.

1403 **5.2.4 Outputs**

1404 None

1405 **5.2.5 Returns**

1406 None

1407 **5.2.6 Example**

```
1408 papi_status_t status;  
1409 papi_service_t handle = NULL;  
1410 ...  
1411 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1412                             "user", "password", NULL,  
1413                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1414 ...  
1415 papiServiceDestroy(handle);
```

1416 **5.2.7 See Also**

1417 [papiServiceCreate](#)

1418 **5.3 *papiServiceSetUserName***

1419 **5.3.1 Description**

1420 Set the user name in the print service handle to be used in subsequent calls. Memory is

1421 allocated and a copy of the input argument is created so that the handle can be used outside
1422 the scope of the input variable.

1423 **5.3.2 Syntax**

```
1424 papi_status_t papiServiceSetUserName( papi_service_t handle,  
1425                                     char* user_name );
```

1426 **5.3.3 Inputs**

1427 **5.3.3.1 handle**

1428 Handle to the print service to update.

1429 **5.3.3.2 user_name**

1430 Points to the name of the user who is making the requests. A NULL value indicates that
1431 the user name associated with the process in which the API call is made should be used.

1432 **5.3.4 Outputs**

1433 **5.3.4.1 handle**

1434 Handle remains unchanged, but it's contents may be updated.

1435 **5.3.5 Returns**

1436 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1437 returned.

1438 **5.3.6 Example**

```
1439 papi_status_t status;  
1440 papi_service_t handle = NULL;  
1441 ...  
1442 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1443                             "user", "password", NULL,  
1444                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1445 ...  
1446 status = papiServiceSetUserName(handle, "root");  
1447 ...  
1448 papiServiceDestroy(handle);
```

1449 **5.3.7 See Also**

1450 [papiServiceCreate](#), [papiServiceGetUserName](#), [papiServiceGetStatusMessage](#)

1451 **5.4 *papiServiceSetPassword***

1452 **5.4.1 Description**

1453 Set the password in the print service handle to be used in subsequent calls. Memory is
1454 allocated and a copy of the input argument is created so that the handle can be used outside
1455 the scope of the input variable.

1456 **5.4.2 Syntax**

```
1457 papi_status_t papiServiceSetPassword( papi_service_t handle, char* password);
```

1458 **5.4.3 Inputs**

1459 **5.4.3.1 *handle***

1460 Handle to the print service to update.

1461 **5.4.3.2 *password***

1462 Points to the password to be used to authenticate the user to the print service.

1463 **5.4.4 Outputs**

1464 **5.4.4.1 *handle***

1465 Handle remains unchanged, but it's contents may be updated.

1466 **5.4.5 Returns**

1467 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1468 returned.

1469 **5.4.6 Example**

```
1470 papi_status_t status;  
1471 papi_service_t handle = NULL;  
1472 ...  
1473 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1474                             "user", "password", NULL,  
1475                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1476 ...  
1477 status = papiServiceSetPassword(handle, "passsword");  
1478 ...  
1479 papiServiceDestroy(handle);
```

1480 **5.4.7 See Also**

1481 [papiServiceCreate](#), [papiServiceGetPassword](#), [papiServiceGetStatusMessage](#)

1482 **5.5 papiServiceSetEncryption**

1483 **5.5.1 Description**

1484 Set the encryption in the print service handle to be used in subsequent calls.

1485 **5.5.2 Syntax**

```
1486 papi_status_t papiServiceSetEncryption( papi_service_t handle,  
1487                                         papi_encryption_t encryption);
```

1488 **5.5.3 Inputs**

1489 **5.5.3.1 handle**

1490 Handle to the print service to update.

1491 **5.5.3.2 encryption**

1492 Specifies the encryption type to be used by the PAPI functions.

1493 **5.5.4 Outputs**

1494 **5.5.4.1 handle**

1495 Handle remains unchanged, but it's contents may be updated.

1496 **5.5.5 Returns**

1497 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1498 returned.

1499 **5.5.6 Example**

```
1500 papi_status_t status;  
1501 papi_service_t handle = NULL;  
1502 ...  
1503 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1504                             "user", "password", NULL,  
1505                             PAPI_ENCRYPT_IF_REQUESTED, NULL);  
1506 ...  
1507 status = papiServiceSetEncryption(handle, PAPI_ENCRYPT_NEVER);  
1508 ...  
1509 papiServiceDestroy(handle);
```

1510 **5.5.7 See Also**

1511 [papiServiceCreate](#), [papiServiceGetEncryption](#), [papiServiceGetStatusMessage](#)

1512 **5.6 papiServiceSetAuthCB**

1513 **5.6.1 Description**

1514 Set the authorization callback function in the print service handle to be used in subsequent
1515 calls.

1516 **5.6.2 Syntax**

```
1517 papi_status_t papiServiceSetAuthCB( papi_service_t handle,  
1518                                     int (*authCB)(papi_service_t svc));
```

1519 **5.6.3 Inputs**

1520 **5.6.3.1 handle**

1521 Handle to the print service to update.

1522 **5.6.3.2 authCB**

1523 Points to a callback function to be used in authenticating the user to the print service if no
1524 password was supplied (or user input is required). A NULL value indicates that no callback
1525 should be made. The callback function should return 0 if the request is to be canceled and
1526 non-zero if new authentication information has been set.

1527 **5.6.4 Outputs**

1528 **5.6.4.1 handle**

1529 Handle remains unchanged, but it's contents may be updated.

1530 **5.6.5 Returns**

1531 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1532 returned.

1533 **5.6.6 Example**

```
1534 papi_status_t status;  
1535 papi_service_t handle = NULL;  
1536 ...  
1537 status = papiServiceCreate(&handle, "ipp://printserver:631",  
1538                             "user", "password", NULL,  
1539                             PAPI_ENCRYPT_IF_REQUESTED, NULL);
```



```
1540 ...
1541 status = papiServiceSetAuthCB(handle, get_password_callback);
1542 ...
1543 papiServiceDestroy(handle);
```

1544 **5.6.7 See Also**

1545 [papiServiceCreate](#), [papiServiceGetStatusMessage](#)

1546 **5.7 papiServiceSetAppData**

1547 **5.7.1 Description**

1548 Set a pointer to some application-specific data in the print service. This data may be used
1549 by the authentication callback function. The caller is responsible for allocating and freeing
1550 memory associated with this data.

1551 **5.7.2 Syntax**

```
1552 papi_status_t papiServiceSetAppData( papi_service_t handle, void *app_data);
```

1553 **5.7.3 Inputs**

1554 **5.7.3.1 handle**

1555 Handle to the print service to update.

1556 **5.7.3.2 app_data**

1557 Points to application-specific data for use by the callback. The caller is responsible for
1558 allocating and freeing memory associated with this data.

1559 **5.7.4 Outputs**

1560 **5.7.4.1 handle**

1561 Handle remains unchanged, but it's contents may be updated.

1562 **5.7.5 Returns**

1563 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1564 returned.

1565 **5.7.6 Example**

```
1566 papi_status_t status;
1567 papi_service_t handle = NULL;
```

Chapter 5: Service API

```
1568 ...
1569 status = papiServiceCreate(&handle, "ipp://printserver:631",
1570                             "user", "password", NULL,
1571                             PAPI_ENCRYPT_IF_REQUESTED, NULL);
1572 ...
1573 status = papiServiceSetAppData(handle, app_data);
1574 ...
1575 papiServiceDestroy(handle);
```

1576 **5.7.7 See Also**

1577 [papiServiceCreate](#), [papiServiceGetAppData](#), [papiServiceGetStatusMessage](#)

1578 **5.8 papiServiceGetServiceName**

1579 **5.8.1 Description**

1580 Get the service name associated with the print service handle.

1581 **5.8.2 Syntax**

```
1582 char *papiServiceGetServiceName(papi_service_t handle);
```

1583 **5.8.3 Inputs**

1584 **5.8.3.1 handle**

1585 Handle to the print service.

1586 **5.8.4 Outputs**

1587 None

1588 **5.8.5 Returns**

1589 A pointer to the service name associated with the print service handle. The value returned
1590 will be deallocated upon destruction of the service handle.

1591 **5.8.6 Example**

```
1592 papi_status_t status;
1593 papi_service_t handle = NULL;
1594 char *service_name = NULL;
1595 ...
1596 service_name = papiServiceGetServiceName(handle);
1597 ...
1598 papiServiceDestroy(handle);
```

1599 **5.8.7 See Also**

1600 [papiServiceCreate](#)

1601 **5.9 papiServiceGetUserName**

1602 **5.9.1 Description**

1603 Get the user name associated with the print service handle.

1604 **5.9.2 Syntax**

```
1605 char *papiServiceGetUserName(papi_service_t handle);
```

1606 **5.9.3 Inputs**

1607 **5.9.3.1 handle**

1608 Handle to the print service.

1609 **5.9.4 Outputs**

1610 None

1611 **5.9.5 Returns**

1612 A pointer to the user name associated with the print service handle.

1613 **5.9.6 Example**

```
1614 papi_status_t status;  
1615 papi_service_t handle = NULL;  
1616 char *service_name = NULL;  
1617 ...  
1618 user_name = papiServiceGetUserName(handle);  
1619 ...  
1620 papiServiceDestroy(handle);
```

1621 **5.9.7 See Also**

1622 [papiServiceCreate](#), [papiServiceSetUserName](#)

1623 **5.10 papiServiceGetPassword**

1624 **5.10.1 Description**

1625 Get the password associated with the print service handle.

1626 **5.10.2 Syntax**

1627 `char *papiServiceGetPassword(papi_service_t handle);`

1628 **5.10.3 Inputs**

1629 **5.10.3.1 handle**

1630 Handle to the print service.

1631 **5.10.4 Outputs**

1632 None

1633 **5.10.5 Returns**

1634 A pointer to the password associated with the print service handle.

1635 **5.10.6 Example**

```
1636 papi_status_t status;  
1637 papi_service_t handle = NULL;  
1638 char *password = NULL;  
1639 ...  
1640 password = papiServiceGetPassword(handle);  
1641 ...  
1642 papiServiceDestroy(handle);
```

1643 **5.10.7 See Also**

1644 [papiServiceCreate](#), [papiServiceSetPassword](#)

1645 **5.11 papiServiceGetEncryption**

1646 **5.11.1 Description**

1647 Get the encryption associated with the print service handle.

1648 **5.11.2 Syntax**

1649 `papi_encryption_t papiServiceGetEncryption(papi_service_t handle);`

1650 **5.11.3 Inputs**

1651 **5.11.3.1 handle**

1652 Handle to the print service.

1653 **5.11.4 Outputs**

1654 None

1655 **5.11.5 Returns**

1656 The type of encryption associated with the print service handle.

1657 **5.11.6 Example**

```
1658 papi_status_t status;  
1659 papi_service_t handle = NULL;  
1660 papi_encryption_t encryption;  
1661 ...  
1662 encryption = papiServiceGetEncryption(handle);  
1663 ...  
1664 papiServiceDestroy(handle);
```

1665 **5.11.7 See Also**

1666 [papiServiceCreate](#), [papiServiceSetEncryption](#)

1667 **5.12 papiServiceGetAppData**

1668 **5.12.1 Description**

1669 Get a pointer to the application-specific data associated with the print service handle.

1670 **5.12.2 Syntax**

```
1671 void *papiServiceGetAppData(papi_service_t handle);
```

1672 **5.12.3 Inputs**

1673 **5.12.3.1 handle**

1674 Handle to the print service.

1675 **5.12.4 Outputs**

1676 None

1677 **5.12.5 Returns**

1678 A pointer to the application-specific data associated with the print service handle.

1679 **5.12.6 Example**

```
1680 papi_status_t status;  
1681 papi_service_t handle = NULL;  
1682 void app_data = NULL;  
1683 ...  
1684 app_data = papiServiceGetAppData(handle);  
1685 ...  
1686 papiServiceDestroy(handle);
```

1687 **5.12.7 See Also**

1688 [papiServiceCreate](#), [papiServiceSetAppData](#)

1689 **5.13 papiServiceGetAttributeList**

1690 **5.13.1 Description**

1691 Retrieve an attribute list from the print service. This attribute list contains service specific
1692 attributes describing service and implementation specific features.

1693 **5.13.2 Syntax**

```
1694 papi_attribute_t **papiServiceGetAttributeList(papi_service_t handle);
```

1695 **5.13.3 Inputs**

1696 **5.13.3.1 handle**

1697 Handle to the print service.

1698 **5.13.4 Outputs**

1699 None

1700 **5.13.5 Returns**

1701 An attribute list associated with the print service handle. The attribute list is destroyed
1702 when the service handle is destroyed.

1703 **5.13.6 Example**

```
1704 papi_status_t status;  
1705 papi_service_t handle = NULL;  
1706 papi_attribute_t **attributes = NULL;  
1707 ...  
1708 attributes = papiServiceGetAttributeList(handle);  
1709 ...
```

```
1710 papiServiceDestroy(handle);
```

1711 **5.13.7 See Also**

1712 [papiServiceCreate](#), [papiServiceDestroy](#)

1713 **5.14 papiServiceGetStatusMessage**

1714 **5.14.1 Description**

1715 Get the message associated with the status of the last operation performed. The status
1716 message returned from this function may be more detailed than the status message returned
1717 from `papiStatusString` (if the print service supports returning more detailed error messages).
1718 The returned message will be localized in the language of the submitter of the original
1719 operation.

1720 **5.14.2 Syntax**

```
1721 Char *papiServiceGetStatusMessage(papi_service_t handle);
```

1722 **5.14.3 Inputs**

1723 **5.14.3.1 handle**

1724 Handle to the print service.

1725 **5.14.4 Outputs**

1726 None

1727 **5.14.5 Returns**

1728 Pointer to the message associated with the print service handle.

1729 **5.14.6 Example**

```
1730 papi_status_t status;  
1731 papi_service_t handle = NULL;  
1732 char *message = NULL;  
1733 ...  
1734 message = papiServiceGetStatusMessage(handle);  
1735 ...  
1736 papiServiceDestroy(handle);
```

1737 **5.14.7 See Also**

1738 [papiServiceCreate](#), [papiServiceSetUserName](#), [papiServiceSetPassword](#),

Chapter 5: Service API

1739 [papiServiceSetEncryption](#), [papiServiceSetAuthCB](#), [papiServiceSetAppData](#), [Printer API](#),
1740 [Attributes API](#), [Job API](#)

1741 **Chapter 6: Printer API**

1742 The printer segment of the PAPI provides a means of interacting with printer objects
1743 contained in a print service. This interaction can include listing, querying, modifying,
1744 pausing, and releasing the printer objects themselves. It can also include clearing all jobs
1745 from a printer object or enumerating all jobs associated with a printer object.

1746 The [papiPrinterQuery](#) function queries all/some of the attributes of a printer object. It
1747 returns a list of printer attributes. A successful call to [papiPrinterQuery](#) is typically followed
1748 by code which examines and processes the returned attributes. When the calling program is
1749 finished with the printer object and its attributes, it should then call [papiPrinterFree](#) to
1750 delete the returned results.

1751 Printers can be found via calls to [papiPrintersList](#). A successful call to [papiPrintersList](#) is
1752 typically followed by code to iterate through the list of returned printers, possibly querying
1753 each ([papiPrinterQuery](#)) for further information (e.g. to restrict what printers get displayed
1754 for a particular user/request). When the calling program is finished with the list of printer
1755 objects, it should then call [papiPrinterListFree](#) to free the returned results.

1756 **6.1 papiPrintersList**

1757 **6.1.1 Description**

1758 List all printers known by the print service which match the specified filter.
1759 Depending on the functionality of the target service's "printer directory", the returned list
1760 may be limited to only printers managed by a particular server or it may include printers
1761 managed by other servers.

1762 **6.1.2 Syntax**

```
1763 papi_status_t papiPrintersList(papi_service_t handle, char *requested_attrs[],  
1764                               papi_filter_t *filter, papi_printer_t **printers );
```

1765 **6.1.3 Inputs**

1766 **6.1.3.1 handle**

1767 Handle to the print service.

1768 **6.1.3.2 requested_attrs**

1769 (optional) NULL terminated array of attributes to be queried. If NULL is passed then all
1770 attributes are queried. (NOTE: The printer may return more attributes than you requested.
1771 This is merely an advisory request that may reduce the amount of data returned if the
1772 printer/server supports it.)

Chapter 6: Printer API

1773 **6.1.3.3 filter**

1774 (optional) Pointer to a filter to limit the number of printers returned on the list request. See
1775 for details. If NULL is passed then all known printers are listed.

1776 **6.1.4 Outputs**

1777 **6.1.4.1 printers**

1778 List of printer objects that matched the filter criteria. The resulting list of printer objects
1779 must be deallocated by the caller using [papiPrinterListFree\(\)](#).

1780 **6.1.5 Returns**

1781 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1782 returned.

1783 **6.1.6 Example**

```
1784 papi_status_t status;  
1785 papi_service_t handle = NULL;  
1786 char *req_attrs[] = { "printer-name", "printer-uri", NULL };  
1787 papi_filter_t filter;  
1788 papi_printer_t *printers = NULL;  
1789 ...  
1790 /* Select local printers (non-remote) that support color */  
1791 filter.type = PAPI_FILTER_BITMASK;  
1792 filter.filter.bitmask.mask = PAPI_PRINTER_REMOTE |  
1793 PAPI_PRINTER_COLOR;  
1794 filter.filter.bitmask.value = PAPI_PRINTER_COLOR;  
1795 ...  
1796 status = papiPrinterList(handle, req_attrs, filter, &printers);  
1797 ...  
1798 if (printers != NULL) {  
1799     int i;  
1800  
1801     for (i = 0; printers[i] != NULL; i++) {  
1802         ...  
1803     }  
1804     papiPrinterListFree(printers);  
1805 }  
1806 ...  
1807 papiServiceDestroy(handle);
```

1808 **6.1.7 See Also**

1809 [papiPrinterListFree](#), [papiPrinterQuery](#)

1810 **6.2 papiPrinterQuery**

1811 **6.2.1 Description**

1812 Queries some or all the attributes of the specified printer object. This includes attributes
1813 representing information and capabilities of the printer. The caller may use this information
1814 to determine which print options to present to the user. How the attributes are obtained (e.g.
1815 from a static database, from a dialog with the hardware, from a dialog with a driver, etc.) is
1816 implementation specific and is beyond the scope of this standard. The call optionally
1817 includes "context" information which specifies job attributes that provide a context that can
1818 be used by the print service to construct capabilities information.

1819 **6.2.2 Semantics Reference**

1820 Get-Printer-Attributes in [RFC2911], section 3.2.5

1821 **6.2.3 Syntax**

```
1822 papi_status_t papiPrinterQuery(papi_service_t handle, char *name,  
1823                               char *requested_attrs[], papi_attribute_t **job_attrs,  
1824                               papi_printer_t *printer );
```

1825 **6.2.4 Inputs**

1826 **6.2.4.1 handle**

1827 Handle to the print service to use.

1828 **6.2.4.2 name**

1829 The name or URI of the printer to query.

1830 **6.2.4.3 requested_attrs**

1831 (optional) NULL terminated array of attributes to be queried. If NULL is passed then all
1832 attributes are queried. (NOTE: The printer may return more attributes than you requested.
1833 This is merely an advisory request that may reduce the amount of data returned if the
1834 printer/server supports it.)

1835 **6.2.4.4 job_attrs**

1836 (optional) NULL terminated array of job attributes in the context of which the capabilities
1837 information is to be constructed. In other words, the returned printer attributes represent the
1838 capabilities of the printer given that these specified job attributes are requested. This allows
1839 for more accurate information to be retrieved by the caller for a specific job (e.g. "if the job
1840 is printed on A4 size media then duplex output is not available"). If NULL is passed then
1841 the full capabilities of the printer are queried.

1842 Support for this argument is optional. If the underlying print system does not have access to
1843 capabilities information bound by job context, then this argument may be ignored. But if
1844 the calling application will be using the returned information to build print job data, then it
1845 is always advisable to specify the job context attributes. The more context information
1846 provided, the more accurate capabilities information is likely to be returned from the print
1847 system.

1848 **6.2.5 Outputs**

1849 **6.2.5.1 printer**

1850 Pointer to a printer object containing the requested attributes.

1851 **6.2.6 Returns**

1852 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1853 returned.

1854 **6.2.7 Example**

```
1855 papi_status_t status;  
1856 papi_service_t handle = NULL;  
1857 char *req_attrs[] = { "printer-name", "printer-uri",  
1858     "printer-state", "printer-state-reasons", NULL };  
1859 papi_attribute_t **job_attrs = NULL;  
1860 papi_printer_t printer = NULL;  
1861 ...  
1862 papiAttributeListAddString(&job_attrs, PAPI_EXCL,  
1863     "media", "legal");  
1864 ...  
1865 status = papiPrinterQuery(handle, "ipp://server/printers/queue",  
1866     req_attrs, job_attrs, &printer);  
1867 papiAttributeListFree(job_attrs);  
1868 ...  
1869 if (printer != NULL) {  
1870     /* process the printer object */  
1871     ...  
1872     papiPrinterFree(printer);  
1873 }  
1874 ...  
1875 papiServiceDestroy(handle);
```

1876 **6.2.8 See Also**

1877 [papiPrintersList](#), [papiPrinterFree](#)

1878 **6.3 *papiPrinterModify***

1879 **6.3.1 Description**

1880 Modifies some or all the attributes of the specified printer object. Upon successful
1881 completion, the function will return a handle to an object representing the updated printer.

1882 **6.3.2 Semantics Reference**

1883 Set-Job-Attributes in [RFC3380], section 4.2

1884 **6.3.3 Syntax**

```
1885 papi_status_t papiPrinterModify(papi_service_t handle, char *printer_name,  
1886 papi_attribute_t **attrs, papi_printer_t *printer );
```

1887 **6.3.4 Inputs**

1888 **6.3.4.1 *handle***

1889 Handle to the print service to use.

1890 **6.3.4.2 *name***

1891 The name or URI of the printer to be modified.

1892 **6.3.4.3 *attrs***

1893 Attributes to be modified. Any attributes not specified are left unchanged. Attributes can be
1894 deleted from the print service's printer object through the use of the PAPI_DELETE
1895 attribute metadata type.

1896 **6.3.5 Outputs**

1897 **6.3.5.1 *printer***

1898 The modified printer object.

1899 **6.3.6 Returns**

1900 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1901 returned.

1902 **6.3.7 Example**

```
1903 papi_status_t status;  
1904 papi_service_t handle = NULL;  
1905 papi_printer_t printer = NULL;
```

Chapter 6: Printer API

```
1906 papi_attribute_t **attrs = NULL;
1907 ...
1908 papiAttributeListAddString(&attrs, PAPI_EXCL,
1909     "printer-location", "Bldg 17/Room 234");
1910 papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
1911     "sample-data", PAPI_DELETE);
1912 ...
1913 status = papiPrinterModify(handle, "printer", attrs, &printer);
1914 ...
1915 if (printer != NULL) {
1916     /* process the printer */
1917     ...
1918     papiPrinterFree(printer);
1919 }
1920 ...
1921 papiServiceDestroy(handle);
```

1922 **6.3.8 See Also**

1923 [PapiPrinterQuery](#), [papiPrinterAdd](#), [papiPrinterRemove](#), [papiPrinterFree](#)

1924 **6.4 papiPrinterAdd**

1925 **6.4.1 Description**

1926 Creates a printer object with some or all the attributes specified. Upon successful
1927 completion, the function will return a handle to an object representing the created printer.

1928 **6.4.2 Semantics Reference**

1929 Set-Job-Attributes in [RFC3380], section 4.2

1930 **6.4.3 Syntax**

```
1931 papi_status_t papiPrinterAdd(papi_service_t handle, char *printer_name,
1932     papi_attribute_t **attrs, papi_printer_t *printer );
```

1933 **6.4.4 Inputs**

1934 **6.4.4.1 handle**

1935 Handle to the print service on which to create the printer object.

1936 **6.4.4.2 name**

1937 The name or URI of the printer to be created

1938 **6.4.4.3 attrs**

1939 Attributes to associate with the printer object being created. The print service may not
1940 honor all requested attributes and it may also add attributes of it's own during printer
1941 creation.

1942 **6.4.5 Outputs**

1943 **6.4.5.1 printer**

1944 The created printer object.

1945 **6.4.6 Returns**

1946 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1947 returned.

1948 **6.4.7 Example**

```
1949 papi_status_t status;  
1950 papi_service_t handle = NULL;  
1951 papi_printer_t printer = NULL;  
1952 papi_attribute_t **attrs = NULL;  
1953 ...  
1954 papiAttributeListAddString(&attrs, PAPI_ATTR_EXCL,  
1955     "printer-location", "Bldg 17/Room 234");  
1956 papiAttributeListAddString(&attrs, PAPI_ATTR_APPEND,  
1957     "document-format-supported", "application/ps");  
1958 papiAttributeListAddString(&attrs, PAPI_ATTR_APPEND,  
1959     "document-format-supported", "text/plain");  
1960 papiAttributeListAddInteger(&attrs, PAPI_ATTR_APPEND,  
1961     "copies-default", 3);  
1962 ...  
1963 status = papiPrinterModify(handle, "ipp://server/printers/triplicate",  
1964     attrs, &printer);  
1965 papiAttributeListFree(attrs);  
1966 ...  
1967 if (status != PAPI_OK) {  
1968     /* report a failure */  
1969 }  
1970  
1971 if (printer != NULL) {  
1972     /* process the printer */  
1973     ...  
1974     papiPrinterFree(printer);  
1975 }  
1976 ...  
1977 papiServiceDestroy(handle);
```

1978 **6.4.8 See Also**

1979 [papiPrinterQuery](#), [papiPrinterModify](#), [papiPrinterRemove](#), [papiPrinterFree](#)

1980 **6.5 papiPrinterRemove**

1981 **6.5.1 Description**

1982 Removes a printer object from a print service.

1983 **6.5.2 Semantics Reference**

1984 Set-Job-Attributes in [RFC3380], section 4.2

1985 **6.5.3 Syntax**

1986 `papi_status_t papiPrinterRemove(papi_service_t handle, char *printer_name);`

1987 **6.5.4 Inputs**

1988 **6.5.4.1 handle**

1989 Handle to the print service from which to remove the printer object.

1990 **6.5.4.2 name**

1991 The name or URI of the printer to be removed

1992 **6.5.5 Outputs**

1993 None

1994 **6.5.6 Returns**

1995 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
1996 returned.

1997 **6.5.7 Example**

```
1998 papi_status_t status;  
1999 papi_service_t handle = NULL;  
2000 ...  
2001 status = papiPrinterRemove(handle, "ipp://server/printers/goodbye");  
2002 if (status != PAPI_OK) {  
2003     /* report a failure */  
2004 }  
2005 ...  
2006 papiServiceDestroy(handle);
```


2007 **6.5.8 See Also**

2008 [papiPrinterQuery](#), [papiPrinterModify](#), [papiPrinterAdd](#), [papiPrinterFree](#)

2009 **6.6 papiPrinterPause**

2010 **6.6.1 Description**

2011 Stops the printer object from scheduling jobs to be printed. Depending on the
2012 implementation, this operation may also stop the printer from processing the current job(s).
2013 This operation is optional and may not be supported by all printers/servers. Use
2014 [papiPrinterResume](#) to undo the effects of this operation.

2015 **6.6.2 Semantics Reference**

2016 Pause-Printer in [RFC2911], section 3.2.7

2017 **6.6.3 Syntax**

2018 `papi_status_t papiPrinterPause(papi_service_t handle, char *name,`
2019 `char *message);`

2020 **6.6.4 Inputs**

2021 **6.6.4.1 handle**

2022 Handle to the print service to use.

2023 **6.6.4.2 name**

2024 The name or URI of the printer to operate on.

2025 **6.6.4.3 message**

2026 (optional) An explanatory message to be associated with the paused printer. This message
2027 may be ignored if the underlying print system does not support associating a message with
2028 a paused printer.

2029 **6.6.5 Outputs**

2030 None

2031 **6.6.6 Returns**

2032 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2033 returned.

2034 **6.6.7 Example**

```
2035 papi_status_t status;  
2036 papi_service_t handle = NULL;  
2037 ...  
2038 status = papiPrinterPause(handle, "printer", "because I can");  
2039 ...  
2040 papiServiceDestroy(handle);
```

2041 **6.6.8 See Also**

2042 [papiPrinterResume](#)

2043 **6.7 papiPrinterResume**

2044 **6.7.1 Description**

2045 Requests that the printer resume scheduling jobs to be printed (i.e. it undoes the effects of
2046 [papiPrinterPause](#)). This operation is optional and may not be supported by all
2047 printers/servers, but it must be supported if [papiPrinterPause](#) is supported.

2048 **6.7.2 Semantics Reference**

2049 Resume-Printer in [RFC2911], section 3.2.8

2050 **6.7.3 Syntax**

```
2051 papi_status_t papiPrinterResume(papi_service_t handle, char *name);
```

2052 **6.7.4 Inputs**

2053 **6.7.4.1 handle**

2054 Handle to the print service to use.

2055 **6.7.4.2 name**

2056 The name or URI of the printer to operate on.

2057 **6.7.5 Outputs**

2058 None

2059 **6.7.6 Returns**

2060 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2061 returned.

2062 **6.7.7 Example**

```
2063 papi_status_t status;  
2064 papi_service_t handle = NULL;  
2065 ...  
2066 status = papiPrinterResume(handle, "printer");  
2067 ...  
2068 papiServiceDestroy(handle);
```

2069 **6.7.8 See Also**

2070 [papiPrinterPause](#)

2071 **6.8 papiPrinterEnable**

2072 **6.8.1 Description**

2073 Requests that the printer enable job creation (queueing) (i.e. it undoes the effects of
2074 [papiPrinterDisable](#)). This operation is optional and may not be supported by all
2075 printers/servers, but it must be supported if [papiPrinterDisable](#) is supported.

2076 **6.8.2 Semantics Reference**

2077 Resume-Printer in [RFC2911], section 3.2.8

2078 **6.8.3 Syntax**

```
2079 papi_status_t papiPrinterEnable(papi_service_t handle, char *name);
```

2080 **6.8.4 Inputs**

2081 **6.8.4.1 handle**

2082 Handle to the print service to use.

2083 **6.8.4.2 name**

2084 The name or URI of the printer to operate on.

2085 **6.8.5 Outputs**

2086 None

2087 **6.8.6 Returns**

2088 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2089 returned.

2090 **6.8.7 Example**

```
2091 papi_status_t status;  
2092 papi_service_t handle = NULL;  
2093 ...  
2094 status = papiPrinterEnable(handle, "printer");  
2095 ...  
2096 papiServiceDestroy(handle);
```

2097 **6.8.8 See Also**

2098 [papiPrinterDisable](#)

2099 **6.9 papiPrinterDisable**

2100 **6.9.1 Description**

2101 Requests that the printer disable job creation (queueing) (i.e. it undoes the effects of
2102 [papiPrinterEnable](#)). This operation is optional and may not be supported by all
2103 printers/servers, but it must be supported if [papiPrinterEnable](#) is supported.

2104 **6.9.2 Semantics Reference**

2105 Resume-Printer in [RFC2911], section 3.2.8

2106 **6.9.3 Syntax**

```
2107 papi_status_t papiPrinterEnable(papi_service_t handle, char *name, char *message);
```

2108 **6.9.4 Inputs**

2109 **6.9.4.1 handle**

2110 Handle to the print service to use.

2111 **6.9.4.2 name**

2112 The name or URI of the printer to operate on.

2113 **6.9.4.3 Message**

2114 An optional reason for disabling the print queue.

2115 **6.9.5 Outputs**

2116 None

2117 **6.9.6 Returns**

2118 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2119 returned.

2120 **6.9.7 Example**

```
2121 papi_status_t status;  
2122 papi_service_t handle = NULL;  
2123 ...  
2124 status = papiPrinterEnable(handle, "printer", "because it's tuesday");  
2125 ...  
2126 papiServiceDestroy(handle);
```

2127 **6.9.8 See Also**

2128 [papiPrinterDisable](#)

2129 **6.10 papiPrinterPurgeJobs**

2130 **6.10.1 Description**

2131 Remove all jobs from the specified printer object regardless of their states. This includes
2132 removing jobs that have completed and are being retained(if any). This operation is optional
2133 and may not be supported by all printers/servers.

2134 **6.10.2 Semantics Reference**

2135 Purge-Jobs in [RFC2911], section 3.2.9

2136 **6.10.3 Syntax**

```
2137 papi_status_t papiPrinterPurgeJobs(papi_service_t handle, char *name,  
2138 papi_job_t **jobs);
```

2139 **6.10.4 Inputs**

2140 **6.10.4.1 handle**

2141 Handle to the print service to use.

2142 **6.10.4.2 name**

2143 The name or URI of the printer to operate on.

2144 **6.10.5 Outputs**2145 **6.10.5.1 jobs**

2146 (optional) Pointer to a list of purged jobs with the identifying information (job-id/job-uri),
 2147 success/fail, and possibly a detailed message. If NULL is passed then no job list is returned.
 2148 Support for the returned job list is optional and may not be supported by all
 2149 implementations (if not supported, the function completes with PAPI_OK_SUBST but no
 2150 list is returned).

2151 **6.10.6 Returns**

2152 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
 2153 returned.

2154 **6.10.7 Example**

```

2155 #include "papi.h"
2156
2157 papi_status_t status;
2158 papi_service_t handle = NULL;
2159 char *service_name = "ipp://printserv:631";
2160 char *user_name = "pappy";
2161 char *password = "goober";
2162 char *printer_name = "my-printer";
2163 papi_job_t *jobs = NULL;
2164
2165 status = papiServiceCreate(handle, service_name, user_name,
2166                          password, NULL, PAPI_ENCRYPT_IF_REQUESTED,
2167                          NULL);
2168 if (status != PAPI_OK) {
2169     /* handle the error */
2170     ...
2171 }
2172
2173 status = papiPrinterPurgeJobs(handle, printer_name, &jobs);
2174 if (status != PAPI_OK) {
2175     /* handle the error */
2176     fprintf(stderr, "papiPrinterPurgeJobs failed: %s\n",
2177            papiServiceGetStatusMessage(handle));
2178     ...
2179 }
2180
2181 if (jobs != NULL) {
2182     int i;
2183
2184     for(i=0; jobs[i] != NULL; i++) {
2185         /* process the job */
2186         ...
2187     }

```

```
2188     papiJobListFree(jobs);
2189 }
2190 ...
2191
2192 papiServiceDestroy(handle);
```

2193 **6.10.8 See Also**

2194 [papiJobCancel](#), [papiJobListFree](#)

2195 **6.11 papiPrinterListJobs**

2196 **6.11.1 Description**

2197 List print job(s) associated with the specified printer.

2198 **6.11.2 Semantics Reference**

2199 Get-Jobs in [RFC2911], section 3.2.6

2200 **6.11.3 Syntax**

```
2201 papi_status_t papiPrinterListJobs(papi_service_t handle, char *printer,
2202                                   char *requested_attrs[], int type_mask,
2203                                   int max_num_jobs, papi_job_t **jobs);
```

2204 **6.11.4 Inputs**

2205 **6.11.4.1 handle**

2206 Handle to the print service to use.

2207 **6.11.4.2 name**

2208 The name or URI of the printer to query.

2209 **6.11.4.3 requested_attrs**

2210 (optional) NULL terminated array of attributes to be queried. If NULL is passed then all
2211 available attributes are queried. (NOTE: The printer may return more attributes than you
2212 requested. This is merely an advisory request that may reduce the amount of data returned
2213 if the printer/server supports it.)

2214 **6.11.4.4 type_mask**

2215 A bit mask which determines what jobs will get returned. The following constants can be
2216 bitwise-OR-ed together to select which types of jobs to list:

Chapter 6: Printer API

```
2217     #define PAPI_LIST_JOBS_OTHERS    0x0001 /* return jobs other than
2218                                     those submitted by the
2219                                     user name associated with
2220                                     the handle */
2221     #define PAPI_LIST_JOBS_COMPLETED  0x0002 /* return completed jobs */
2222     #define PAPI_LIST_JOBS_NOT_COMPLETED 0x0004 /* return not-completed
2223                                     jobs */
2224     #define PAPI_LIST_JOBS_ALL       0xFFFF /* return all jobs */
```

2225 **6.11.4.5 max_num_jobs**

2226 Limit to the number of jobs returned. If 0 is passed, then there is no limit to the number of
2227 jobs which may be returned.

2228 **6.11.5 Outputs**

2229 **6.11.5.1 jobs**

2230 List of job objects returned.

2231 **6.11.6 Returns**

2232 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2233 returned.

2234 **6.11.7 Example**

```
2235 papi_status_t status;
2236 papi_service_t handle = NULL;
2237 papi_job_t *jobs = NULL;
2238 char *job_attrs[] = {
2239     "job-id", "job-name", "job-originating-user-name",
2240     "job-state", "job-state-reasons", "job-state-message" };
2241 ...
2242 status = papiPrinterListJobs(handle, printer_name, job_attrs,
2243                             PAPI_LIST_JOBS_ALL, 0, &jobs);
2244 ...
2245 if (jobs != NULL) {
2246     int i;
2247
2248     for(i = 0; jobs[i] != NULL; i++) {
2249         /* process the job */
2250         ...
2251     }
2252     papiJobListFree(jobs);
2253 }
2254 ...
2255 papiServiceDestroy(handle);
```


2256 **6.11.8 See Also**

2257 [papiJobQuery](#), [papiJobListFree](#)

2258 **6.12 *papiPrinterGetAttributeList***

2259 **6.12.1 Description**

2260 Get the attribute list associated with a printer object.

2261 This function retrieves an attribute list from a printer object returned in a previous call.

2262 Printer objects are returned as the result of operations performed by [papiPrintersList](#),

2263 [papiPrinterQuery](#), and [papiPrinterModify](#).

2264 **6.12.2 Syntax**

```
2265 papi_attribute_t **papiPrinterGetAttributeList(papi_printer_t printer);
```

2266 **6.12.3 Inputs**

2267 **6.12.3.1 *printer***

2268 Handle of the printer object.

2269 **6.12.4 Outputs**

2270 none

2271 **6.12.5 Returns**

2272 Pointer to the attribute list associated with the printer object. This attribute list is

2273 deallocated when the printer object it was retrieved from is deallocated using

2274 [papiPrinterFree](#)(printer).

2275 **6.12.6 Example**

```
2276 papi_attribute_t **attrs = NULL;
2277 papi_printer_t printer = NULL;
2278 ...
2279 attrs = papiPrinterGetAttributeList(printer);
2280 ...
2281 papiPrinterFree(printer);
```

2282 **6.12.7 See Also**

2283 [papiPrintersList](#), [papiPrinterQuery](#), [papiPrinterModify](#)

2284 **6.13 *papiPrinterFree***

2285 **6.13.1 Description**

2286 Free a printer object.

2287 **6.13.2 Syntax**

```
2288 void papiPrinterFree(papi_printer_t printer);
```

2289 **6.13.3 Inputs**

2290 **6.13.3.1 *printer***

2291 Handle of the printer object to free.

2292 **6.13.4 Outputs**

2293 none

2294 **6.13.5 Returns**

2295 none

2296 **6.13.6 Example**

```
2297 papi_printer_t printer = NULL;  
2298 ...  
2299 papiPrinterFree(printer);
```

2300 **6.13.7 See Also**

2301 [papiPrinterQuery](#), [papiPrinterModify](#)

2302 **6.14 *papiPrinterListFree***

2303 **6.14.1 Description**

2304 Free a list of printer objects.

2305 **6.14.2 Syntax**

```
2306 void papiPrinterListFree(papi_printer_t *printers);
```

2307 **6.14.3 Inputs**

2308 **6.14.3.1 printers**

2309 Pointer to the printer object list to free.

2310 **6.14.4 Outputs**

2311 none

2312 **6.14.5 Returns**

2313 none

2314 **6.14.6 Example**

```
2315 papi_printer_t* printers = NULL;  
2316 ...  
2317 papiPrinterListFree (printers);
```

2318 **6.14.7 See Also**

2319 [papiPrintersList](#)

2320 **Chapter 7: Job API**

2321 The job segment of the PAPI provides a means of interacting with job objects contained in
2322 a print service. This interaction can include listing, querying, creating, modifying,
2323 canceling, holding, releasing, and restarting the job objects themselves.

2324 The [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobStreamOpen](#), and
2325 [papiJobStreamClose](#) functions provide a means of creating job objects under a print service.
2326 The [papiJobValidate](#) function can be used to determine if a job submission will be
2327 successful. Each of these functions results in a job object with an attribute list that can be
2328 queried to determine what the resulting job looks like.

2329 The [papiJobQuery](#) function queries all/some of the attributes of a job. A successful call to
2330 [papiJobQuery](#) is typically followed by code which examines and processes the returned
2331 attributes. When the calling program is finished with the job object and it's attributes, it
2332 should then call [papiJobFree](#) to delete the returned results.

2333 Jobs and job state can be modified through the use of [papiJobModify](#), [papiJobHold](#),
2334 [papiJobRelease](#), and [papiJobRestart](#). The [papiJobModify](#) call returns a job object that
2335 contains a representation of the modified job. The job object's attribute list can be queried
2336 to determin what the resulting job looks like. When the calling program is finished with the
2337 job object and it's attributes, it should then call [papiJobFree](#) to delete the returned results.

2338 **7.1 *papiJobSubmit***

2339 **7.1.1 Description**

2340 Submits a print job having the specified attributes to the specified printer. This interface
2341 copies the specified print files before returning to the caller (contrast to
2342 [papiJobSubmitByReference](#)). The caller must call [papiJobFree](#) when done in order to free
2343 the resources associated with the returned job object. Attributes of the print job may be
2344 passed in the `job_attributes` argument and/or in a job ticket (using the `job_ticket` argument).
2345 If both are specified, the attributes in the `job_attributes` list will be applied to the `job_ticket`
2346 attributes and the resulting attribute set will be used.

2347 **7.1.2 Semantics Reference**

2348 Print-Job in [RFC2911], section 3.2.1

2349 **7.1.3 Syntax**

```
2350 papi_status_t papiJobSubmit(papi_service_t handle, char *printer_name,  
2351                             papi_attribute_t **job_attributes,  
2352                             papi_job_ticket_t *job_ticket,  
2353                             char **file_names, papi_job_t *job );
```

2354 **7.1.4 Inputs**

2355 **7.1.4.1 handle**

2356 Handle to the print service to use.

2357 **7.1.4.2 printer_name**

2358 Pointer to the name of the printer to which the job is to be submitted.

2359 **7.1.4.3 job_attributes**

2360 (optional) The list of attributes describing the job and how it is to be printed. If options are
2361 specified here and also in the job ticket data, the value specified here takes precedence. If
2362 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2363 job.

2364 **7.1.4.4 job_ticket**

2365 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2366 job ticket is used with the job. Whether the implementation passes both the attributes and
2367 the job ticket to the server/printer, or merges them to some print protocol or internal
2368 representation depends on the implementation.

2369 **7.1.4.5 file_names**

2370 NULL terminated list of pointers to names of files to print. If more than one file is
2371 specified, the files will be treated by the print system as separate "documents" for things
2372 like page breaks and separator sheets, but they will be scheduled and printed together as one
2373 job and the specified attributes will apply to all the files.
2374 These file names may contain absolute path names or relative path names (relative to the
2375 current path). The implementation MUST copy the file contents before returning.

2376 **7.1.5 Outputs**

2377 **7.1.5.1 job**

2378 The resulting job object representing the submitted job. The caller must deallocate this
2379 object using [papiJobFree\(\)](#) when finished using it.

2380 **7.1.6 Returns**

2381 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2382 returned.

2383 **7.1.7 Example**

2384 `papi_status_t status;`

Chapter 7: Job API

```
2385 papi_service_t handle = NULL;
2386 papi_attribute_t **attrs = NULL;
2387 papi_job_ticket_t *ticket = NULL;
2388 char *files[] = { "/etc/motd", NULL };
2389 papi_job_t job = NULL;
2390 ...
2391 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
2392                             PAPI_STRING, 1, "test job");
2393 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
2394                              PAPI_INTEGER, 4);
2395 ...
2396 status = papiJobSubmit(handle, "printer", attrs, ticket, files, &job);
2397 papiAttributeListFree(attrs);
2398 ...
2399 if (job != NULL) {
2400     /* look at the job object (maybe get the id) */
2401     papiJobFree(job);
2402 }
2403 ...
```

2404 **7.1.8 See Also**

2405 [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamOpen](#), [papiJobStreamWrite](#),
2406 [papiJobStreamClose](#), [papiJobFree](#)

2407 **7.2 papiJobSubmitByReference**

2408 **7.2.1 Description**

2409 Submits a print job having the specified attributes to the specified printer. This interface
2410 delays copying the specified print files as long as possible, ideally only "pulling" the files
2411 when the printer is actually printing the job (contrast to [papiJobSubmit](#)).
2412 Attributes of the print job may be passed in the `job_attributes` argument and/or in a job
2413 ticket (using the `job_ticket` argument). If both are specified, the attributes in the
2414 `job_attributes` list will be applied to the `job_ticket` attributes and the resulting attribute set
2415 will be used.

2416 **7.2.2 Semantics Reference**

2417 Print-URI in [RFC2911], section 3.2.2

2418 **7.2.3 Syntax**

```
2419 papi_status_t papiJobSubmitByReference(papi_service_t handle,
2420                                       char *printer_name,
2421                                       papi_attribute_t **job_attributes,
2422                                       papi_job_ticket_t *job_ticket,
```

2423 `char **file_names, papi_job_t *job);`

2424 **7.2.4 Inputs**

2425 **7.2.4.1 handle**

2426 Handle to the print service to use.

2427 **7.2.4.2 printer_name**

2428 Pointer to the name of the printer to which the job is to be submitted.

2429 **7.2.4.3 job_attributes**

2430 (optional) The list of attributes describing the job and how it is to be printed. If options are
2431 specified here and also in the job ticket data, the value specified here takes precedence. If
2432 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2433 job.

2434 **7.2.4.4 job_ticket**

2435 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2436 job ticket is used with the job. Whether the implementation passes both the attributes and
2437 the job ticket to the server/printer, or merges them to some print protocol or internal
2438 representation depends on the implementation.

2439 **7.2.4.5 file_names**

2440 NULL terminated list of pointers to names of files to print. If more than one file is
2441 specified, the files will be treated by the print system as separate "documents" for things
2442 like page breaks and separator sheets, but they will be scheduled and printed together as one
2443 job and the specified attributes will apply to all the files.

2444 These file names may contain absolute path names, relative path names or URIs
2445 ([RFC1738], [RFC2396]). The implementation SHOULD NOT copy the referenced data
2446 unless (or until) it is no longer feasible to maintain the reference. Feasibility limitations
2447 may arise out of security issues, name space issues, and/or protocol or printer limitations.
2448 Implementations MUST support the absolute path, relative path, and "file:" URI scheme.
2449 Use of other URI schemes could result in a PAPI_URI_SCHEME error, depending on the
2450 implementation.

2451 The semantics explained in the preceding paragraphs allows for flexibility in the PAPI
2452 implementation. For example: (1) PAPI on top of a local service to maintain the reference
2453 for the life of the job, if the local service supports it. (2) PAPI on top of IPP to send a
2454 reference when the server can access the referenced data and copy it when it is not
2455 accessible to the server. (3) PAPI on top of network printing protocols that don't support
2456 references to copy the data on the way out to the remote server.

2457 **7.2.5 Outputs**

2458 **7.2.5.1 job**

2459 The resulting job object representing the submitted job. The caller must deallocate this
2460 object using [papiJobFree\(\)](#) when finished using it.
2461

2462 **7.2.6 Returns**

2463 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2464 returned.

2465 **7.2.7 Example**

```
2466 papi_status_t status;  
2467 papi_service_t handle = NULL;  
2468 papi_attribute_t **attrs = NULL;  
2469 papi_job_ticket_t *ticket = NULL;  
2470 char *files[] = { "/etc/motd", NULL };  
2471 papi_job_t job = NULL;  
2472 ...  
2473 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,  
2474                             PAPI_STRING, 1, "test job");  
2475 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,  
2476                              PAPI_INTEGER, 4);  
2477  
2478 status = papiJobSubmitByReference(handle, "printer", attrs, ticket,  
2479                                  files, &job);  
2480 papiAttributeListFree(attrs)  
2481 ...  
2482 if (job != NULL) {  
2483     /* look at the job object (maybe get the id) */  
2484     papiJobFree(job);  
2485 }  
2486 ...
```

2487 **7.2.8 See Also**

2488 [papiJobSubmit](#), [papiJobValidate](#), [papiJobStreamOpen](#), [papiJobStreamWrite](#),
2489 [papiJobStreamClose](#), [papiJobFree](#)

2490 **7.3 papiJobValidate**

2491 **7.3.1 Description**

2492 Validates the specified job attributes against the specified printer. This function can be used
2493 to validate the capability of a print object to accept a specific combination of attributes.
2494 Attributes of the print job may be passed in the job_attributes argument and/or in a job

2495 ticket (using the `job_ticket` argument). If both are specified, the attributes in the
2496 `job_attributes` list will be applied to the `job_ticket` attributes and the resulting attribute set
2497 will be used.

2498 **7.3.2 Semantics Reference**

2499 Validate-Job in [RFC2911], section 3.2.3

2500 **7.3.3 Syntax**

```
2501 papi_status_t papiJobValidate(papi_service_t handle, char *printer_name,  
2502                             papi_attribute_t **job_attributes,  
2503                             papi_job_ticket_t *job_ticket,  
2504                             char **file_names, papi_job_t *job );
```

2505 **7.3.4 Inputs**

2506 **7.3.4.1 *handle***

2507 Handle to the print service to use.

2508 **7.3.4.2 *printer_name***

2509 Pointer to the name of the printer to which the job is to be validated.

2510 **7.3.4.3 *job_attributes***

2511 (optional) The list of attributes describing the job and how it is to be printed. If options are
2512 specified here and also in the job ticket data, the value specified here takes precedence. If
2513 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2514 job.

2515 **7.3.4.4 *job_ticket***

2516 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2517 job ticket is used with the job. Whether the implementation passes both the attributes and
2518 the job ticket to the server/printer, or merges them to some print protocol or internal
2519 representation depends on the implementation.

2520 **7.3.4.5 *file_names***

2521 NULL terminated list of pointers to names of files to validate.

2522 **7.3.5 Outputs**

2523 **7.3.5.1 *job***

2524 The resulting job object representing the validated job. The caller must deallocate this

Chapter 7: Job API

2525 object using [papiJobFree\(\)](#) when finished using it.

2526

2527 **7.3.6 Returns**

2528 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2529 returned.

2530 **7.3.7 Example**

```
2531 papi_status_t status;  
2532 papi_service_t handle = NULL;  
2533 papi_attribute_t **attrs = NULL;  
2534 papi_job_ticket_t *ticket = NULL;  
2535 char *files[] = { "/etc/motd", NULL };  
2536 papi_job_t job = NULL;  
2537 ...  
2538 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,  
2539                             PAPI_STRING, 1, "test job");  
2540 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,  
2541                              PAPI_INTEGER, 4);  
2542 ...  
2543 status = papiJobValidate(handle, printer, attrs, ticket, files, &job);  
2544 papiAttributeListFree(attrs);  
2545 ...  
2546 if (job != NULL) {  
2547     papiJobFree(job);  
2548 }  
2549 ...
```

2550 **7.3.8 See Also**

2551 [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobStreamOpen](#), [papiJobStreamWrite](#),
2552 [papiJobStreamClose](#), [papiJobFree](#)

2553 **7.4 *papiJobStreamOpen***

2554 **7.4.1 Description**

2555 Opens a print job and an associated stream of print data to be sent to the specified printer.
2556 After calling this function [papiJobStreamWrite](#) can be called (repeatedly) to write the print
2557 data to the stream, and then [papiJobStreamClose](#) is called to complete the submission of the
2558 print job.

2559 After this function is called successfully, [papiJobStreamClose](#) must eventually be called to
2560 close the stream (this includes all error paths).

2561 Attributes of the print job may be passed in the job_attributes argument and/or in a job
2562 ticket (using the job_ticket argument). If both are specified, the attributes in the
2563 job_attributes list will be applied to the job_ticket attributes and the resulting attribute set

2564 will be used.

2565 **7.4.2 Syntax**

```
2566 papi_status_t papiJobStreamOpen(papi_service_t handle, char *printer_name,  
2567                               papi_attribute_t **job_attributes,  
2568                               papi_job_ticket_t *job_ticket,  
2569                               papi_stream_t *stream);
```

2570 **7.4.3 Inputs**

2571 **7.4.3.1 handle**

2572 Handle to the print service to use.

2573 **7.4.3.2 printer_name**

2574 Pointer to the name of the printer to which the job is to be validated.

2575 **7.4.3.3 job_attributes**

2576 (optional) The list of attributes describing the job and how it is to be printed. If options are
2577 specified here and also in the job ticket data, the value specified here takes precedence. If
2578 this is NULL then only default attributes and (optionally) a job ticket is submitted with the
2579 job.

2580 **7.4.3.4 job_ticket**

2581 (optional) Pointer to structure specifying the job ticket. If this argument is NULL, then no
2582 job ticket is used with the job. Whether the implementation passes both the attributes and
2583 the job ticket to the server/printer, or merges them to some print protocol or internal
2584 representation depends on the implementation.

2585 **7.4.4 Outputs**

2586 **7.4.4.1 stream**

2587 The resulting stream object to which print data can be written. The stream object will be
2588 deallocated when closed using [papiJobStreamClose\(\)](#).
2589

2590 **7.4.5 Returns**

2591 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2592 returned.

2593 7.4.6 Example

```
2594 papi_status_t status;
2595 papi_service_t handle = NULL;
2596 papi_attribute_t **attrs = NULL;
2597 papi_job_ticket_t *ticket = NULL;
2598 papi_job_t job = NULL;
2599 char buffer[4096];
2600 size_t buflen = 0;
2601 ...
2602 papiAttributeListAddString(attrs, "job-name", PAPI_ATTR_EXCL,
2603                             PAPI_STRING, 1, "test job");
2604 papiAttributeListAddInteger(attrs, "copies", PAPI_ATTR_EXCL,
2605                              PAPI_INTEGER, 4);
2606 ...
2607 status = papiJobStreamOpen(handle, "printer", attrs, ticket, &stream);
2608 papiAttributeListFree(attrs);
2609 ...
2610 while (print_data_remaining) {
2611     status = papiJobStreamWrite(handle, stream, buffer, buflen);
2612 }
2613 ...
2614 status = papiJobStreamClose(handle, stream, &job);
2615 ...
2616 if (job != NULL) {
2617     ...
2618     papiJobFree(job);
2619 }
2620 ...
```

2621 7.4.7 See Also

2622 [papiJobStreamWrite](#), [papiJobStreamClose](#)

2623 7.5 *papiJobStreamWrite*

2624 7.5.1 Description

2625 Writes print data to the specified open job stream. The open job stream must have been
2626 obtained by a successful call to [papiJobStreamOpen](#)

2627 7.5.2 Syntax

```
2628 papi_status_t papiJobStreamWrite(papi_service_t handle, papi_stream_t stream,
2629                                 void *buffer, size_t buflen);
```

2630 **7.5.3 Inputs**

2631 **7.5.3.1 handle**

2632 Handle to the print service to use.

2633 **7.5.3.2 stream**

2634 The open stream object to which print data is written.

2635 **7.5.3.3 buffer**

2636 Pointer to the buffer of print data to write.

2637 **7.5.3.4 buflen**

2638 The number of bytes to write.

2639 **7.5.4 Outputs**

2640 none

2641 **7.5.5 Returns**

2642 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2643 returned.

2644 **7.5.6 Example**

2645 See `papiJobStreamOpen`

2646 **7.5.7 See Also**

2647 [papiJobStreamOpen](#), [papiJobStreamClose](#)

2648 **7.6 *papiJobStreamClose***

2649 **7.6.1 Description**

2650 Closes the specified open job stream and completes submission of the job (if there were no
2651 previous errors returned from `papiJobSubmitWrite`). The open job stream must have been
2652 obtained by a successful call to `papiJobStreamOpen`.

2653 **7.6.2 Syntax**

```
2654 papi_status_t papiJobStreamClose(papi_service_t handle, papi_stream_t stream,  
2655                                 papi_job_t *job);
```

2656 **7.6.3 Inputs**

2657 **7.6.3.1 handle**

2658 Handle to the print service to use.

2659 **7.6.3.2 stream**

2660 The open stream object to close.

2661 **7.6.4 Outputs**

2662 **7.6.4.1 Job**

2663 The resulting job object representing the submitted job. The caller must deallocate this
2664 object using [papiJobFree\(\)](#) when finished using it.

2665 **7.6.5 Returns**

2666 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2667 returned.

2668 **7.6.6 Example**

2669 See `papiJobStreamOpen`

2670 **7.6.7 See Also**

2671 [papiJobStreamOpen](#), [papiJobStreamWrite](#)

2672 **7.7 papiJobQuery**

2673 **7.7.1 Description**

2674 Queries some or all the attributes of the specified job object.

2675 **7.7.2 Semantics Reference**

2676 Get-Job-Attributes in [RFC2911], section 3.3.4

2677 **7.7.3 Syntax**

```
2678 papi_status_t papiJobQuery(papi_service_t handle, char* printer_name,  
2679                          int32_t job_id, char *requested_attrs[],  
2680                          papi_job_t *job);
```

2681 **7.7.4 Inputs**

2682 **7.7.4.1 handle**

2683 Handle to the print service to use.

2684 **7.7.4.2 printer_name**

2685 Pointer to the name or URI of the printer to which the job was submitted.

2686 **7.7.4.3 job_id**

2687 The ID number of the job to be queried.

2688 **7.7.4.4 requested_attrs**

2689 NULL terminated array of attributes to be queried. If NULL is passed then all available
2690 attributes are queried. (NOTE: The job may return more attributes than you requested. This
2691 is merely an advisory request that may reduce the amount of data returned if the
2692 printer/server supports it.)

2693 **7.7.5 Outputs**

2694 **7.7.5.1 job**

2695 The returned job object containing the requested attributes.

2696 **7.7.6 Returns**

2697 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2698 returned.

2699 **7.7.7 Example**

```
2700 papi_status_t status;  
2701 papi_service_t handle = NULL;  
2702 papi_job_t job = NULL;  
2703 char *job_attrs[] = {  
2704     "job-id", "job-name", "job-originating-user-name",  
2705     "job-state", "job-state-reasons", NULL };  
2706 ...  
2707 status = papiJobQuery(handle, "printer", job_id, job_attrs, &job);  
2708 ...  
2709 if (job != NULL) {  
2710     /* process the job */  
2711     ...  
2712     papiJobFree(job);  
2713 }  
2714 ...
```

2715 **7.7.8 See Also**

2716 [papiPrinterListJobs](#), [papiJobFree](#)

2717 **7.8 papiJobModify**

2718 **7.8.1 Description**

2719 Modifies some or all the attributes of the specified job object. Upon successful completion,
2720 the function will return a handle to an object representing the updated job.

2721 **7.8.2 Semantics Reference**

2722 Set-Job-Attributes in [RFC3380], section 4.2

2723 **7.8.3 Syntax**

```
2724 papi_status_t papiJobModify(papi_service_t handle, char* printer_name,  
2725                             int32_t job_id, papi_attribute_t **attrs,  
2726                             papi_job_t *job);
```

2727 **7.8.4 Inputs**

2728 **7.8.4.1 handle**

2729 Handle to the print service to use.

2730 **7.8.4.2 printer_name**

2731 Pointer to the name or URI of the printer to which the job was submitted.

2732 **7.8.4.3 job_id**

2733 The ID number of the job to be queried.

2734 **7.8.4.4 attrs**

2735 Attributes to be modified. Any attributes not specified are left unchanged.

2736 **7.8.5 Outputs**

2737 **7.8.5.1 job**

2738 The modified job object.

2739 **7.8.6 Returns**

2740 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2741 returned.

2742 **7.8.7 Example**

```
2743 papi_status_t status;
2744 papi_service_t handle = NULL;
2745 papi_job_t job = NULL;
2746 papi_attribute_t **attrs = NULL;
2747 ...
2748 papiAttributeListAddString(&attrs, PAPI_EXCL,
2749                             "job-name", "sample job");
2750 papiAttributeListAddMetadata(&attrs, PAPI_EXCL,
2751                               "media", PAPI_DELETE);
2752 ...
2753 status = papiJobModify(handle, "printer", 12, attrs, &job);
2754 ...
2755 if (job != NULL) {
2756     /* process the job */
2757     ...
2758     papiJobFree(job);
2759 }
2760 ...
```

2761 **7.8.8 See Also**

2762 [papiJobFree](#)

2763 **7.9 *papiJobCancel***

2764 **7.9.1 Description**

2765 Cancel the specified print job

2766 **7.9.2 Semantics Reference**

2767 Cancel Job in [RFC2911], section 3.3.3

2768 **7.9.3 Syntax**

```
2769 papi_status_t papiJobCancel(papi_service_t handle, char* printer_name,
2770                             int32_t job_id);
```

2771 **7.9.4 Inputs**

2772 **7.9.4.1 *handle***

2773 Handle to the print service to use.

2774 **7.9.4.2 *printer_name***

2775 Pointer to the name or URI of the printer to which the job was submitted.

2776 **7.9.4.3 *job_id***

2777 The ID number of the job to be canceled.

2778 **7.9.5 Outputs**

2779 none

2780 **7.9.6 Returns**

2781 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2782 returned.

2783 **7.9.7 Example**

```
2784 papi_status_t status;  
2785 papi_service_t handle = NULL;  
2786 ...  
2787 status = papiJobCancel(handle, "printer", 12);  
2788 ...
```

2789 **7.9.8 See Also**

2790 [papiPrinterPurgeJobs](#)

2791 **7.10 *papiJobHold***

2792 **7.10.1 Description**

2793 Hold the specified print job

2794 **7.10.2 Semantics Reference**

2795 Hold Job in [RFC2911], section 3.3.5

2796 **7.10.3 Syntax**

```
2797 papi_status_t papiJobHold(papi_service_t handle, char* printer_name,  
2798                          int32_t job_id);
```

2799 **7.10.4 Inputs**

2800 **7.10.4.1 *handle***

2801 Handle to the print service to use.

2802 **7.10.4.2 printer_name**

2803 Pointer to the name or URI of the printer to which the job was submitted.

2804 **7.10.4.3 job_id**

2805 The ID number of the job to be held.

2806 **7.10.5 Outputs**

2807 none

2808 **7.10.6 Returns**

2809 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2810 returned.

2811 **7.10.7 Example**

```
2812 papi_status_t status;  
2813 papi_service_t handle = NULL;  
2814 ...  
2815 status = papiJobHold(handle, "printer", 12);  
2816 ...
```

2817 **7.10.8 See Also**

2818 [papiJobRelease](#)

2819 **7.11 papiJobRelease**

2820 **7.11.1 Description**

2821 Release the specified print job

2822 **7.11.2 Semantics Reference**

2823 Release Job in [RFC2911], section 3.3.6

2824 **7.11.3 Syntax**

```
2825 papi_status_t papiJobRelease(papi_service_t handle, char* printer_name,  
2826                             int32_t job_id);
```

2827 **7.11.4 Inputs**

2828 **7.11.4.1 handle**

2829 Handle to the print service to use.

2830 **7.11.4.2 printer_name**

2831 Pointer to the name or URI of the printer to which the job was submitted.

2832 **7.11.4.3 job_id**

2833 The ID number of the job to be released.

2834 **7.11.5 Outputs**

2835 none

2836 **7.11.6 Returns**

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

2839 **7.11.7 Example**

```
2840 papi_status_t status;  
2841 papi_service_t handle = NULL;  
2842 ...  
2843 status = papiJobRelease(handle, "printer", 12);  
2844 ...
```

2845 **7.11.8 See Also**

2846 [papiJobHold](#)

2847 **7.12 papiJobRestart**

2848 **7.12.1 Description**

2849 Restarts a job that was retained after processing. If and how a job is retained after
2850 processing is implementation-specific and is not covered by this API. This operation is
2851 optional and may not be supported by all printers/servers.

2852 **7.12.2 Semantics Reference**

2853 Restart Job in [RFC2911], section 3.3.7

2854 **7.12.3 Syntax**

```
2855 papi_status_t papiJobRestart(papi_service_t handle, char* printer_name,  
2856 int32_t job_id);
```

2857 **7.12.4 Inputs**

2858 **7.12.4.1 handle**

2859 Handle to the print service to use.

2860 **7.12.4.2 printer_name**

2861 Pointer to the name or URI of the printer to which the job was submitted.

2862 **7.12.4.3 job_id**

2863 The ID number of the job to be restart.

2864 **7.12.5 Outputs**

2865 none

2866 **7.12.6 Returns**

2867 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2868 returned.

2869 **7.12.7 Example**

```
2870 papi_status_t status;  
2871 papi_service_t handle = NULL;  
2872 ...  
2873 status = papiJobRestart(handle, "printer", 12);  
2874 ...
```

2875 **7.12.8 See Also**

2876 [papiJobHold](#), [papiJobRelease](#)

2877 **7.13 papiJobPromote**

2878 **7.13.1 Description**

2879 Promotes a job to the front of the queue so that it may be printed after any currently
2880 printing job completes. This operation is optional and may not be supported by all
2881 printers/servers.

2882 **7.13.2 Semantics Reference**

2883 Restart Job in [RFC2911], section 3.3.7

2884 **7.13.3 Syntax**

```
2885 papi_status_t papiJobPromote(papi_service_t handle,char* printer_name,  
2886                             int32_t job_id);
```

2887 **7.13.4 Inputs**

2888 **7.13.4.1 handle**

2889 Handle to the print service to use.

2890 **7.13.4.2 printer_name**

2891 Pointer to the name or URI of the printer to which the job was submitted.

2892 **7.13.4.3 job_id**

2893 The ID number of the job to be promoted.

2894 **7.13.5 Outputs**

2895 none

2896 **7.13.6 Returns**

2897 If successful, a value of PAPI_OK is returned. Otherwise an appropriate failure value is
2898 returned.

2899 **7.13.7 Example**

```
2900 papi_status_t status;  
2901 papi_service_t handle = NULL;  
2902 ...  
2903 status = papiJobPromote(handle, "printer", 12);  
2904 ...
```

2905 **7.13.8 See Also**

2906 [papiJobHold](#), [papiJobRelease](#)

2907 **7.14 *papiJobGetAttributeList***

2908 **7.14.1 Description**

2909 Get the attribute list associated with a job object.
2910 This function retrieves an attribute list from a job object returned in a previous call. Job
2911 objects are returned as a result of the operations performed by [papiPrinterListJobs](#),
2912 [papiJobQuery](#), [papiJobModify](#), [papiJobSubmit](#), [papiJobSubmitByReference](#),
2913 [papiJobValidate](#), and [papiJobStreamClose](#).

2914 **7.14.2 Syntax.**

```
2915 papi_attribute_t **papiJobGetAttributeList(papi_job_t job);
```

2916 **7.14.3 Inputs**

2917 **7.14.3.1 *job***

2918 Handle of the job object.

2919 **7.14.4 Outputs**

2920 none

2921 **7.14.5 Returns**

2922 The attribute list associated with the job object. The attribute list is deallocated when the
2923 containing job object is destroyed using [papiJobFree\(\)](#).

2924 **7.14.6 Example**

```
2925 papi_job_t job = NULL;  
2926 papi_attribute_list **attrs = NULL;  
2927 ...  
2928 attrs = papiJobGetAttributeList(job);  
2929 ...  
2930 papiJobFree(job);
```

2932 **7.14.7 See Also**

2933 [papiPrinterListJobs](#), [papiJobQuery](#), [papiJobModify](#), [papiJobSubmit](#),
2934 [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamClose](#)

2935 **7.15 *papiJobGetPrinterName***

2936 **7.15.1 Description**

2937 Get the printer name associated with a job object.

2938 **7.15.2 Syntax.**

```
2939 char *papiJobGetPrinterName(papi_job_t job);
```

2940 **7.15.3 Inputs**

2941 **7.15.3.1 *job***

2942 Handle of the job object.

2943 **7.15.4 Outputs**

2944 none

2945 **7.15.5 Returns**

2946 Pointer to the printer name associated with the job object. The resulting string is
2947 deallocated when the containing job object is destroyed using [papiJobFree\(\)](#).

2948 **7.15.6 Example**

```
2949 char *printer = NULL;  
2950 papi_job_t job = NULL;  
2951 ...  
2952 printer = papiJobGetPrinterName(job);  
2953 ...  
2954 papiJobFree(job);
```

2956 **7.15.7 See Also**

2957 [papiJobGetAttributeList](#)

2958 **7.16 *papiJobGetId***

2959 **7.16.1 Description**

2960 Get the job ID associated with a job object.

2961 **7.16.2 Syntax.**

2962 `int32_t papiJobGetId(papi_job_t job);`

2963 **7.16.3 Inputs**

2964 **7.16.3.1 job**

2965 Handle of the job object.

2966 **7.16.4 Outputs**

2967 none

2968 **7.16.5 Returns**

2969 The job id associated with the job object.

2970 **7.16.6 Example**

```
2971 papi_job_t job = NULL;
2972 int32_t id;
2973 ...
2974 id = papiJobGetId(job);
2975 ...
2976 papiJobFree(job);
```

2977 **7.16.7 See Also**

2978 [papiJobGetAttributeList](#)

2979 **7.17 papiJobGetJobTicket**

2980 **7.17.1 Description**

2981 Get the job ticket associated with a job object. The job ticket is deallocated when the
2982 containing job object is destroyed using [papiJobFree\(\)](#).

2983 **7.17.2 Syntax**

2984 `papi_job_ticket_t *papiJobGetJobTicket(papi_job_t job);`

2985 **7.17.3 Inputs**

2986 **7.17.3.1 job**

2987 Handle of the job object.

2988 **7.17.4 Outputs**

2989 none

2990 **7.17.5 Returns**

2991 Pointer to the job ticket associated with the job object.

2992 **7.17.6 Example**

```
2993 papi_job_t job = NULL;  
2994 papi_job_ticket_t *ticket;  
2995 ...  
2996 ticket = papiJobGetJobTicket(job);  
2997 ...  
2998 papiJobFree(job);
```

2999 **7.17.7 See Also**

3000 [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamOpen](#)

3001 **7.18 *papiJobFree***

3002 **7.18.1 Description**

3003 Free a job object.

3004 **7.18.2 Syntax**

```
3005 void papiJobFree(papi_job_t job);
```

3006 **7.18.3 Inputs**

3007 **7.18.3.1 Job**

3008 Handle of the job object to free.

3009 **7.18.4 Outputs**

3010 none

3011 **7.18.5 Returns**

3012 none

3013 **7.18.6 Example**

```
3014 papi_job_t job = NULL;
```

```
3015 ...
3016 papiJobFree ( job ) ;
```

3017 **7.18.7 See Also**

3018 [papiJobSubmit](#), [papiJobSubmitByReference](#), [papiJobValidate](#), [papiJobStreamClose](#),
3019 [papiJobQuery](#), [papiJobModify](#)

3020 **7.19 papiJobListFree**

3021 **7.19.1 Description**

3022 Free a job list.

3023 **7.19.2 Syntax**

```
3024 void papiJobListFree(papi_job_t *job );
```

3025 **7.19.3 Inputs**

3026 **7.19.3.1 Job**

3027 Handle of the job list to free.

3028 **7.19.4 Outputs**

3029 none

3030 **7.19.5 Returns**

3031 none

3032 **7.19.6 Example**

```
3033 papi_job_t *jobs = NULL;
3034 ...
3035 papiJobListFree ( jobs ) ;
```

3036 **7.19.7 See Also**

3037 [papiPrinterListJobs](#)

3038 **Chapter 8: Miscellaneous API**

3039 **8.1 *papiStatusString***

3040 **8.1.1 Description**

3041 Get a status string for the specified `papi_status_t`. The status message returned from this
3042 function may be less detailed than the status message returned from
3043 [papiServiceGetStatusMessage](#) (if the print service supports returning more detailed error
3044 messages)

3045 **8.1.2 Syntax**

```
3046 char *papiStatusString(papi_status_t status);
```

3047 **8.1.3 Inputs**

3048 **8.1.3.1 *status***

3049 The status value to convert to a status string

3050 **8.1.4 Outputs**

3051 none

3052 **8.1.5 Returns**

3053 The returned string provides a (potentially localized) human readable message representing
3054 the status provided. The return value should not be deallocated by the caller.

3055 **8.1.6 Example**

```
3056 papi_status_t status;  
3057 char *message;  
3058 ...  
3059 message = papiServiceGetStatusMessage(handle);  
3060 ...
```

3061 **8.1.7 See Also**

3062 [PapiServiceGetStatusMessage](#)

3063 **8.2 *papiLibrarySupportedCalls***

3064 **8.2.1 Description**

3065 The `papiLibrarySupportedCalls()` function can be called to request a list of API functions

3066 that are supported in the implementation. Support for a function means that the
3067 implementation of that function is not a stub that simply returns
3068 PAPI_OPERATION_NOT_SUPPORTED

3069 **8.2.2 Syntax**

```
3070 char **papiLibrarySupportedCalls();
```

3071 **8.2.3 Inputs**

3072 none

3073 **8.2.4 Outputs**

3074 none

3075 **8.2.5 Returns**

3076 A NULL terminated list of supported function names. This list should not be deallocated
3077 by the caller.

3078 **8.2.6 Example**

```
3079 papi_service_t handle = NULL;  
3080 char **calls;  
3081 ...  
3082 calls = papiLibrarySupportedCalls(handle);  
3083 ...
```

3084 **8.2.7 See Also**

3085 Conformance Table

3086 **8.3 *papiLibrarySupportedCall***

3087 **8.3.1 Description**

3088 The papiLibrarySupportedCalls() function can be called to request a list of API functions
3089 that are supported in the implementation. Support for a function means that the
3090 implementation of that function is not a stub that simply returns
3091 PAPI_OPERATION_NOT_SUPPORTED

3092 **8.3.2 Syntax**

```
3093 char papiLibrarySupportedCall(char *name);
```

3094 **8.3.3 Inputs**

3095 **8.3.3.1 name**

3096 The name of the function that is being asked about

3097 **8.3.4 Outputs**

3098 none

3099 **8.3.5 Returns**

3100 A return of PAPI_TRUE indicates that the named function is supported by the API
3101 implementation. A return of PAPI_FALSE indicates that the the named function is not
3102 supported by the API implementation.

3103 **8.3.6 Example**

```
3104 papi_service_t handle = NULL;  
3105 char supported;  
3106 ...  
3107 supported = papiLibrarySupportedCall(handle, "papiJobQuery");  
3108 ...
```

3109 **8.3.7 See Also**

3110 [ConformanceProfiles](#)

3111 **Chapter 9: Capabilities**

3112 **9.1 Introduction**

3113 In the context of this document, printer capabilities refers to information about the features,
3114 options, limitation, etc. of a print device (either an actual device or an abstract device which
3115 may represent a group or pool of actual devices). This includes such information as:

- 3116 • Does the printer support color printing?
- 3117 • At what resolution(s) can the printer print?
- 3118 • What input trays are present?
- 3119 • What size media is loaded in each tray?
- 3120 • Which trays are manual-feed and which are auto-feed?
- 3121 • Can the printer print duplex output?
- 3122 • What is the printable area on each of the loaded media?
- 3123 • What output bins are present?
- 3124 • What finishing (staple, punch, etc.) does the printer support?
- 3125 • What combinations of features are not allowed together?
- 3126 • What features should be presented on the print user interface?
- 3127 • ... and many others...

3128 The uses of printer capabilities by applications include:

3129 To control how to display print options in a print UI dialog. Examples:

- 3130 • What values to put in the binselection pull-down lists
- 3131 • Whether or not to gray-out the duplex option when a particular output bin
3132 has been selected.
- 3133 • Whether or not to display a color vs. black and white selection

3134 To Control how the printdata stream is generated. Examples:

- 3135 • How large an image to draw to fill the printable area.
- 3136 • How much to shift the image if “3-hole punch” finishing has been
3137 selected.
- 3138 • How to request that the printer print on paper from the manual envelope
3139 feeder

3140 To do job validation and printer selection. Examples:

- 3141 • Can I print this job with these options on this printer?

- 3142 • Find a printer which can print this job with these options.

3143 **9.2 Objectives**

3144 This section attempts to describe the objectives of the PAPI printer capabilities support. It
3145 is important to understand these objectives in order to understand why the support is
3146 structured the way that it is.

3147 **9.2.1 Standard printer capabilities API**

3148 There is no standard API which a Linux application can use to retrieve printer capabilities
3149 regardless of the device, driver, and print server being used. This makes it very difficult for
3150 application writers to support generating print data without writing multiple versions of the
3151 print logic or without tying the application to very specific print system environments. This
3152 specification provides the standard API, making applications which use it independent of
3153 the underlying print system.

3154 **9.2.2 Independent of underlying source of capabilities**

3155 The capabilities information returned to the application may come from one of a variety of
3156 sources or combination of sources. The data retrieved from these sources may be
3157 represented in a variety of formats, including:

- 3158 • PPD files
- 3159 • UPDF database
- 3160 • SNMP queries
- 3161 • Device drivers

3162 The API defined here hides these differences so that the application is independent of data
3163 source and format used.

3164 **9.2.3 Support returning information in context**

3165 The API supports a means for requesting capabilities information in the context of a
3166 particular set of job options. For example, set of printer capabilities can be queried given
3167 that medium and color/black-and-white selections have already been made.

3168 **9.2.4 Support returning constraints**

3169 The API must support a means for returning constraints on printer capabilities. This allows
3170 applications to not submit job with disallowed combinations of options, and to display
3171 better print job dialogs (gray-out potentially conflicting options, highlight conflicting
3172 options that have been selected, display an error message when invalid combinations are
3173 submitted, etc.).

3174 The constraints returned should allow some level of “boolean logic”, including

3175 negation, to simplify the information returned. For example, to not allow doing finishing
3176 when transparencies are selected as the medium, it would be preferable if the constraints
3177 could express “(type – transparency) AND (finishing NOT = none)” instead of having to
3178 list a combination of “(type = transparency)” with every possible finishing value other than
3179 “none”.

3180 **9.2.5 Support returning display hints**

3181 The API should support a means of returning “display hints”. This is information that the
3182 application can use to display print options in a print dialog that is easy to use. For
3183 example, returning information about which options should be displayed on the “main
3184 window”, which should be displayed in an “advanced” dialog, and which should not be
3185 displayed at all.

3186 **9.2.6 Support logically grouping features**

3187 The API should support a means of returning logical groupings of printer features. This is
3188 information about combinations of lower-level features that can be displayed and selected
3189 as a group to make the user interface easier to use. For example, a group of features called
3190 “black-and-white-draft” could include a logical setting of the color, resolution, and print
3191 density options.

3192 The feature group support should be an open, extendible way for printer vendors and print
3193 administrators to express logical and commonly used groupings of print options that make it
3194 easier for end-users to take advantage of lower-level printer features. They should not be
3195 used to blindly list all possible combinations of a set of options, whether or not all the
3196 combinations make sense.

3197 **9.3 Interfaces**

3198 **9.3.1 Query Functionality**

3199 The API used by the application to retrieve printer capabilities is the [apiPrinterQuery](#)
3200 function. See the description of that function for further details.

3201 **9.3.2 Capability Attributes**

3202 In addition to the xxx-supported attributes defined by the IPP standard [RFC2911], this
3203 section defines new attributes needed to satisfy the objectives described above.

3204 **9.3.2.1 Job-constraints-col (1 setOf collection)**

3205 Constraints are expressed in the printer object's job-constraints-col attribute. This attribute
3206 is multi-valued with each value having collection syntax. Each value is, in fact, an attribute
3207 list that represents one combination of job attributes/values which are not allowed for that
3208 printer. If an attribute in the collection does not have a value, then all values of that
3209 attribute are disallowed in this combination.

Chapter 9: Capabilities

3210 The set of values associated with job-constraints-col represents the complete set of job
3211 attribute constraints associated with the containing printer object.

3212 The attribute values in job-constraints-col may also be in range syntax, if the corresponding
3213 job attribute has integer syntax. This represents the included (or excluded, if the attribute is
3214 named in job-constraints-inverted) range of values of that attribute within that constraint.

3215 **9.3.2.2 Job-constraints-inverted (1setOf type2 keyword)**

3216 The job-constraints-inverted attribute lists the names of other attributes in the current job-
3217 constraints-col value whose comparison logic must be inverted. That is, the values of
3218 named attributes are to be excluded (“not equal to” values) from the constraint. If an
3219 attribute name is not included in the job-constraints-inverted attribute, then that attribute's
3220 values are to be included (“equal to” values) in the constraint.

3221 You can think of the each attribute in a job-constraints-cols value as AND-ed together to
3222 express a disallowed combination of options: “(attr1 == values) AND (attr2 == values)
3223 AND ...”. The job constraints-inverted attribute lists those attribute/value comparisons
3224 which are to be “!=” instead of “==”.

3225 **9.3.3 Example**

3226 Here is an example of how the job-constraints-col attribute can be used to express various
3227 printer constraints. The example is expressed in pseudo-code with curly brackets enclosing
3228 each collection value and attributes within each collection are shown on separate lines with
3229 commas separating the values (this is the PAPI text encoding format documented in
3230 [Chapter 11: Attribute List Text Representation](#), with the addition of not-legal-syntax
3231 comments in “/* ... */” to help describe the examples):

```
3232 job-constraints-col = {  
3233     /*  
3234     * Constraint: no high print quality with 240 dpi resolution  
3235     * (print-quality == high) AND (printer-resolution == 240dpi)  
3236     */  
3237     {  
3238         print-quality = high  
3239         printer -resolution = 240dpi  
3240     },  
3241     /*  
3242     * Constraint: no transparency with duplex  
3243     * (sides != one-sided) AND (media – transparency)  
3244     */  
3245     {  
3246         job-constraints-inverted = sides  
3247         sides = one-sided  
3248     }
```

```

3249     media = transparency
3250 },
3251
3252 /*
3253  * Constraint: no finishing with heavy-stock media
3254  * (finishings != none) AND (media == heavy-stock)
3255  /
3256 {
3257     job-constraints-inverted = finishing
3258     finishings = none
3259     media = heavy-stock
3260 },
3261
3262 /*
3263  * Constraint: no duplex printing of A4 paper in landscape
3264  * (sides != one-sided) AND (media == A4) AND
3265  * (orientation-requested == landscape)
3266  /
3267 {
3268     job-constraints-inverted = sides
3269     sides = one-sided
3270     media = A4
3271     orientation-requested = landscape
3272 },
3273
3274 /*
3275  * Constraint: no duplex printing of COM-10 envelopes
3276  * (sides != one-sided) AND (media == envelope) AND
3277  * (media-size == com10)
3278  /
3279 {
3280     job-constraints-inverted = sides
3281     sides = one-sided
3282     media = envelope
3283     media-size = com10
3284 },
3285
3286 /*
3287  * Constraint: no stapling of greater than 50 sheets
3288  * (finishings == staple) AND (job-media-sheets > 50)
3289  */
3290 {
3291     job-constraints-inverted = job-media-sheets

```

Chapter 9: Capabilities

```
3292     finishings = staple
3293     job-media-sheets = 1-50
3294 }
3295 };
```

3296 **9.3.4 Validation Function**

3297 The API used by the application to validate print job attributes against printer capabilities is
3298 the [papiJobValidate](#) function. See the description of that function for further details.

3299 **Chapter 10: Attributes**

3300 For a summary of the IPP attributes which can be used with the PAPI interface, see:
3301 <ftp://ftp.pwg.org/pub/pwg/fsg/spool/IPP-Object-Attributes.pdf>

3302 **10.1 Extension Attributes**

3303 The following attributes are not currently defined by IPP, but may be used with this API.

3304 **10.1.1 Job-ticket-formats-supported**

3305 (1setOf type2 keyword) This optional printer attribute lists the job ticket formats that are
3306 supported by the printer. If this attribute is not present, it is assumed that the printer does
3307 not support any job ticket formats

3308 **10.1.2 media-margins**

3309 (1setOf integer) The media-margins attribute defines the printable margins for the current
3310 printer object and consists of exactly 4 or 8 ordered integers. Each group of 4 integers
3311 represent the minimum distance from the top, right, bottom, and left edges of the media in
3312 100ths of millimeters.

3313 If 4 integers are provided, the margins are the same for the front and back sides of the
3314 media when producing duplexed output. If 8 integers are provided, the first 4 integers
3315 represent the margins for the front side and the last 4 integers represent the margins for the
3316 back side of the media.

3317 Currently the margin values only represent the minimum margins that can be used with all
3318 sizes and types of media. Future versions of the PAPI specification may define an interface
3319 for getting the margin values for specific combinations of job template attributes.

3320 **10.2 Required Job Attributes**

3321 The following job attributes must be supported to comply with this API standard. These
3322 attributes may be supported by the underlying print server directly, or they may be mapped
3323 by the PAPI library.

- 3324 • job-id
- 3325 • job-name
- 3326 • job-originating-user-name
- 3327 • job-printer-uri
- 3328 • job-state
- 3329 • job-state-reasons
- 3330 • job-uri

Chapter 10: Attributes

- 3331 • time-at-creation
- 3332 • time-at-processing
- 3333 • time-at-completed

3334 **10.3 Required Printer Attributes**

3335 The following printer attributes must be supported to comply with this API standard. These
3336 attributes may be supported by the underlying print server directly, or they may be mapped
3337 by the PAPI library.

- 3338 • charset-configured
- 3339 • charset-supported
- 3340 • compression-supported
- 3341 • document-format-default
- 3342 • document-format-supported
- 3343 • generated-natural-language-supported
- 3344 • natural-language-configured
- 3345 • operations-supported
- 3346 • pdl-override-supported
- 3347 • printer-is-accepting-jobs
- 3348 • printer-name
- 3349 • printer-state
- 3350 • printer-state-reasons
- 3351 • printer-up-time
- 3352 • printer-uri-supported
- 3353 • queued-job-count
- 3354 • uri-authentication-supported
- 3355 • uri-security-supported

3356 **10.4 IPP Attribute Type Mapping**

3357 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)

3358

3359 **Chapter 11: Attribute List Text Representation**3360 **11.1 ABNF Definition**

3361 The following ABNF definition [RFC2234] describes the syntax of PAPI attributes
 3362 encoded as text options:

```

3363 OPTION-STRING = [OPTION] *(1*WC OPTION) *WC
3364
3365 OPTION        = TRUEOPTION / FALSEOPTION / VALUEOPTION
3366
3367 TRUEOPTION    = NAME
3368
3369 FALSEOPTION   = "no" NAME
3370
3371 VALUEOPTION   = NAME "=" VALUE *( "," VALUE )
3372
3373 NAME          = 1*NAMECHAR
3374
3375 NAMECHAR     = DIGIT / ALPHA / "-" / "_" / "."
3376
3377 VALUE        = BOOLVALUE / COLVALUE / DATEVALUE / NUMBERVALUE /
3378 QUOTEDVALUE /
3379             RANGEVALUE / RESVALUE / STRINGVALUE
3380
3381 BOOLVALUE    = "yes" / "no" / "true" / "false"
3382
3383 COLVALUE     = "{" OPTION-STRING "}"
3384
3385 DATEVALUE    = HOUR MINUTE [ SECOND ] / YEAR MONTH DAY /
3386             YEAR MONTH DAY HOUR MINUTE [ SECOND ]
3387
3388 YEAR         = 4DIGIT
3389
3390 MONTH        = "0" %x31-39 / "10" / "11" / "12"
3391
3392 DAY          = %x30-32 DIGIT / "1" DIGIT / "2" DIGIT / "30" / "31"
3393
3394 HOUR         = %x30-31 DIGIT / "1" DIGIT / "20" / "21" / "22" / "23"
3395
3396 MINUTE       = %x30-35 DIGIT
3397
3398 SECOND       = %x30-35 DIGIT
3399
3400 NUMBERVALUE  = 1*DIGIT / "-" 1*DIGIT / "+" 1*DIGIT
3401
3402 QUOTEDVALUE  = DQUOTE *QUOTEDCHAR DQUOTE / SQUOTE *QUOTEDCHAR SQUOTE
3403
3404 QUOTEDCHAR   = %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
3405             %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
3406             %x5D-7E / %xA0-FF

```



```

3407
3408 OCTALDIGIT      = %x30-37
3409
3410 RANGEVALUE     = 1*DIGIT "-" 1*DIGIT
3411
3412 RESVALUE       = 1*DIGIT [ "x" 1*DIGIT ] ("dpi" / "dpc")
3413
3414 STRINGVALUE    = 1*STRINGCHAR
3415
3416 STRINGCHAR     = %x5C %x20 / %x5C %x5C / %x5C DQUOTE / %x5C SQUOTE /
3417                 %x5C 3OCTALDIGIT / %x21 / %x23-26 / %x28-5B /
3418                 %x5D-7E / %xA0-FF
3419
3420 SQUOTE        = %x27
3421
3422 WC            = %x09 / %x0A / %x20

```

3423 **11.2 Examples**

3424 The following example strings illustrate the format of text options:

3425 **11.2.1 Boolean Attributes**

```

3426 foo
3427 nofoo
3428 foo=false
3429 foo=true
3430 foo=no
3431 foo=yes
3432

```

3433

3434 **11.2.2 Collection Attributes**

```

3435 media-col={media-size={x-dimension=123 y-dimension=456}}
3436

```

3437

3438 **11.2.3 Integer Attributes**

```

3439 copies=123
3440 hue=-123
3441

```

3442

3443 **11.2.4 String Attributes**

```
3444 job-sheets=standard
3445 job-sheets=standard,standard
3446 media=na-custom-foo.8000-10000
3447 job-name=John\'s\ Really\040Nice\ Document
3448
```

3449

3450 **11.2.5 String Attributes (quoted)**

```
3451 job-name="John\'s Really Nice Document"
3452 document-name='Another \'Word\042 document.doc'
3453
```

3454

3455 **11.2.6 Range Attributes**

```
3456 page-ranges=1-5
3457 page-ranges=1-2,5-6,101-120
3458
```

3459

3460 **11.2.7 Date Attributes**

```
3461 job-hold-until-datetime=1234
3462 job-hold-until-datetime=123456
3463 job-hold-until-datetime=20020904
3464 job-hold-until-datetime=200209041234
3465 job-hold-until-datetime=20020904123456
3466
```

3467

3468 **11.2.8 Resolution Attributes**

```
3469 resolution=360dpi
3470 resolution=720x360dpi
3471 resolution=1000dpc
3472
```

3473

3474 **11.2.9 Multiple Attributes**

```
3475 job-sheets=standard page-ranges=1-2,5-6,101-120 resolution=360dpi
```

3476 Chapter 12: Conformance

3477 There are some cases where it may not be necessary or even desirable to implement the
3478 interfaces defined in this specification in their entirety. This section describes which
3479 elements of the interfaces must be implemented and defines sets of interfaces that may be
3480 implemented. The sets of interfaces that may be implemented define various levels of
3481 conformance. Conformance to a particular level may only be claimed by an
3482 implementation if and only if all of the interfaces defined in that level are implemented as
3483 described in their associated section of this document. These implementations may only
3484 return PAPI_OPERATION_NOT_SUPPORTED if and only if the underlying support has
3485 been administratively disabled. Regardless of conformance level claimed by an
3486 implementation, the header file for every implementation must be complete. That is to say
3487 that it must include a complete set of type definitions, enumeration and function prototypes.

3488 12.1 Query Profile

3489 The Query Profile is defined to provide querying functionality. A PAPI implementation
3490 conforming to the Query Profile must provide code for all functions defined in the PAPI
3491 and must support all of the definitions in the “papi.h” C header file. For each function
3492 defined in the PAPI specification, a conforming implementation must either perform the
3493 requested function or return the PAPI_OPERATION_NOT_SUPPORTED status code (see
3494 section 3.8). The PAPI_OPERATION_NOT_SUPPORTED status code indicates either (1)
3495 that the PAPI implementation doesn’t provide any support for the function, i.e., the function
3496 is stubbed out, or (2), the PAPI implementation does provide *code support* for the function,
3497 but the Printer or Print system selected by the application does not support the
3498 corresponding function.

3499 lists the functions and attributes that a PAPI implementation is REQUIRED to provide
3500 *code support* in order to claim conformance to the Query Profile. The blank entries are
3501 OPTIONAL for a PAPI implementation to support.

3503 12.2 Job Submission Profile

3504 The Job Submission Profile is defined to provide the job submission functionality and is a
3505 superset of the Querying Profile. lists the functions and attributes that a PAPI
3506 implementation is REQUIRED to provide *code support* in order to claim conformance to
3507 the Job Submission Profile. The blank entries are OPTIONAL for a PAPI implementation
3508 to support.

3509 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 papiAttributeListAdd	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

3510

3511 **Chapter 13: Sample Code**

3512 Sample implementations of this specification and client applications built upon it can be
3513 found at <http://sf.net/projects/OpenPrinting/> While the implemenations and clients
3514 applications found there are intended to be true to the spec, they are not authoritative. This
3515 document is the athoritavie definition of the Free Standard Group Open Standard Print API
3516 (PAPI).

3517 **Chapter 14: References**

3518 **14.1 Internet Printing Protocol (IPP)**

3519 IETF RFCs can be obtained from "<http://www.rfc-editor.org/rfcsearch.html>". Other IPP
3520 documents can be obtained from "<http://www.pwg.org/ipp/index.html>" and
3521 "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

3522 **14.2 Job Ticket**

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

3523 **14.3 Printer Working Group (PWG)**

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

3524 **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)

3525 **Chapter 15: Change History**

3526 **15.1 Version 1.0 (May 9, 2005)**

3527 Added note about interfaces to be covered in the next release.

3528 **15.2 Version 0.92 (January 12, 2005).**

3529 Added administrative operations: papiPrinterAdd, papiPrinterRemove, papiPrinter Enable,
3530 papiPrinterDisable, papiJobPromote.

3531 **15.3 Version 0.91 (January 28, 2004).**

3532 Pruned several example code excerpts to the essential information required to get a better
3533 understanding of the various calls.

3534 Added/modified introductory text for Attribute, Service, Printer, and Job API chapters.

3535 Added papi_metadata_t type/support for various OOB IPP types that we need to support.

3536 Converted from SGML to OpenOffice to be able to use versioning, change bars, line
3537 number, ... (will begin using versioning and change bars after this release)

3538 Added numerous cross references.

3539 Added papiLibrarySupportedCall() and papiLibrarySupportedCalls(). To enumerate/verify
3540 actual support for a function in the library

3541 Added papiServiceGetAttributeList() call to retrieve print service and implementation
3542 specific information from a service handle.

3543 Added a "Conformance" section to the document. A draft introduction and conformance
3544 table are included, but the actual conformance levels need work. The bulk of this was
3545 included from Ira's and Tom's draft.

3546 Moved Attribute section in front of the Service, Printer, and Job sections interfaces to
3547 improve flow of document.

3548 Added papi_encryption_t to common structures

3549 Added constraints chapter. The bulk of this chapter was copied directly from v0.3 of the
3550 papi capabilities document.

3551 **15.4 Version 0.9 (November 18, 2002).**

3552 Changed media-margins order to "top, right, bottom, left" to match other standards.

3553 Changed media-margins units to "100ths of millimeters" to match other standards. Also,
3554 reworded last paragraph of description of this attribute.

Chapter 15: Change History

3555 **15.5 Version 0.8 (November 15, 2002).**

- 3556 Added value field, explanation, and corrected example for `papi_filter_t`.
- 3557 Added `media-margins` attribute to "Extension Attributes" section.
- 3558 Renamed function names with "Username" to "UserName", and renamed function names with "Servicename" to "ServiceName", and Miscellaneous wording and typo corrections.

3560 **15.6 Version 0.7 (October 18, 2002).**

- 3561 Added `attr_delim` argument to `papiAttributeListToString` and made new-line ("`\n`") an
- 3562 allowed attribute delimiter on input to `papiAttributeListFromString`.
- 3563 Added "Semantics Reference" subsections to functions.
- 3564 Added to References: [5101.1], [RFC3196], and URIs for obtaining IPP documents.
- 3565 Added `PAPI_JOB_TICKET_NOT_SUPPORTED` status code.
- 3566 Added "Globalization" section in the "Print System Model" chapter.
- 3567 Changed definition and usage of returned value from `papiAttributeListGetValue`. Also
- 3568 clarified what happens to output values when a `papiAttributeListGet*` call has an error.
- 3569 Clarified descriptions of `papiPrinterGetAttributeList` and `papiJobGetAttributeList`.
- 3570 Changed buffer length arguments from `int` to `size_t`.
- 3571 Clarified that `papiServiceDestroy` must always be called after a call to `papiServiceCreate`.
- 3572 Removed `attributes-charset`, `attributes-natural-language`, and `job-printer-up-time` from the
- 3573 "Required Job Attributes" (they may be hidden inside the PAPI implementation).
- 3574 Clarified result of passing both attributes and a job ticket on all the job submission
- 3575 functions.
- 3576 Miscellaneous wording and typo corrections.

3577 **15.7 Version 0.6 (September 20, 2002)**

- 3578 Made explanation of `requested_attrs` in `papiPrintersList` the same as it is for
- 3579 `papiPrinterQuery`.
- 3580 Moved `units` argument on `papiAttributeListAddResolution` to the end of the argument list to
- 3581 match the corresponding get function.
- 3582 Added `papiAttributeListAddCollection` and `papiAttributeListGetCollection`.
- 3583 Removed unneeded extra level of indirection from `attrs` argument to `papiAttributeListGet*`
- 3584 functions. Also made the `attrs` argument `const`.
- 3585 Added note to "Conventions" section that strings are assumed to be UTF-8 encoded.
- 3586 Added `papiAttributeListFromString` and `papiAttributeListToString` functions, along with a

- 3587 new appendix defining the string format syntax.
- 3588 Added papiJobSubmitByReference, papiJobStreamOpen, papiJobStreamWrite, and
3589 papiJobStreamClose functions.
- 3590 Added short "Document" section in the "Print System Model" chapter.
- 3591 Added explanation of how multiple files specified in the papiJobSubmit file_names
3592 argument are handled by the print system.
- 3593 Changed papi_job_ticket_t "uri" field to "file_name" and added explanation text.
- 3594 Added explanation of implementation option for merging papiJobSubmit attributes with
3595 job_ticket argument.
- 3596 Added "References" appendix.
- 3597 Added "IPP Attribute Type Mapping" appendix.
- 3598 Added "PWG" job ticket format to papi_jt_format_t.
- 3599 Miscellaneous wording and typo corrections.

- 3600 **15.8 Version 0.5 (August 30, 2002).**
- 3601 Added job_attrs argument to papiPrinterQuery to support more accurate query of printer
3602 capabilities.
- 3603 Added management functions papiAttributeDelete, papiJobModify, and papiPrinterModify.
- 3604 Added functions papiAttributeListGetValue, papiAttributeListGetString,
3605 papiAttributeListGetInteger, etc.
- 3606 Renamed papiAttributeAdd* functions to papiAttributeListAdd* to be consistent with the
3607 naming convention (first word after "papi" is the object being operated upon).
- 3608 Changed last argument of papiAttributeListAdd to papi_attribute_value_t*.
- 3609 Made description of authentication more implementation-independent.
- 3610 Added reference to IPP attributes summary document.
- 3611 Added result argument to papiPrinterPurgeJobs.
- 3612 Added "collection attribute" support (PAPI_COLLECTION type).
- 3613 Changed boolean values to consistently use char. Added PAPI_FALSE and PAPI_TRUE
3614 enum values.

- 3615 **15.9 Version 0.4 (July 19, 2002).**
- 3616 Made papi_job_t and papi_printer_t opaque handles and added "get" functions to access the
3617 associated information (papiPrinterGetAttributeList, papiJobGetAttributeList,
3618 papiJobGetId, papiJobGetPrinterName, papiJobGetJobTicket).

Chapter 15: Change History

- 3619 Removed variable length argument lists from attribute add functions.
- 3620 Changed order and name of flag value passed to attribute add functions.
- 3621 Eliminated indirection in date/time value passed to papiAttributeAddDatetime.
- 3622 Added message argument to papiPrinterPause.
- 3623 **15.10 Version 0.3 (June 24, 2002).**
- 3624 Converted to DocBook format from Microsoft Word
- 3625 Major rewrite, including:
- 3626 Changed how printer names are described in "Model/Printer"
- 3627 Changed fixed length strings to pointers in numerous structures/sections
- 3628 Redefined attribute/value structures and associated API descriptions
- 3629 Changed list/query functions to return "objects"
- 3630 Rewrote "Attributes API" chapter
- 3631 Changed many function definitions to pass NULL-terminated arrays of pointers instead of a
- 3632 separate count argument
- 3633 Changed papiJobSubmit to take an attribute list structure as input instead of a formatted
- 3634 string
- 3635 **15.11 Version 0.2 (April 17, 2002).**
- 3636 Updated references to IPP RFC from 2566 (IPP 1.0) to 2911 (IPP 1.1)
- 3637 Filled in "Encryption" section and added information about encryption in "Object
- 3638 Identification" section
- 3639 Added "short_name" field in "Object Identification" section
- 3640 Added "Job Ticket (papi_job_ticket_t)" section
- 3641 Added papiPrinterPause
- 3642 Added papiPrinterResume
- 3643 Added papiPurgeJobs
- 3644 Added optional job_ticket argument to papiJobSubmit
- 3645 Added optional passing of filenames by URI to papiJobSubmit
- 3646 Added papiHoldJob
- 3647 Added papiReleaseJob
- 3648 Added papiRestartJob

3649 **15.12 Version 0.1 (April 3, 2002).**

3650 Original draft version