

Robert Herriot (editor)
Sun Microsystems
Sylvan Butler
Hewlett-Packard
Paul Moore
Microsoft.
Randy Turner
Sharp Labs
July 30+4, 1997

Internet Printing Protocol/1.0: Protocol Specification
draft-ietf-ipp-protocol-010.txt

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "Iid-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

This document is one of a set of documents, which together describe all aspects of a new Internet Printing Protocol (IPP). IPP is an application level protocol that can be used for distributed printing using Internet tools and technology. The protocol is heavily influenced by the printing model introduced in the Document Printing Application (ISO/IEC 10175 DPA) standard. Although DPA specifies both end user and administrative features, IPP version 1.0 is focused only on end user functionality.

The full set of IPP documents includes:

- Internet Printing Protocol: Requirements
- Internet Printing Protocol/1.0: Model and Semantics
- Internet Printing Protocol/1.0: Security
- Internet Printing Protocol/1.0: Protocol Specification
- Internet Printing Protocol/1.0: Directory Schema

The requirements document takes a broad look at distributed printing functionality, and it enumerates real-life scenarios that help to clarify the features that need to be included in a printing protocol for the Internet. It identifies requirements for three types of users: end users, operators, and administrators. The requirements document calls out a subset of end user requirements that MUST be satisfied in the first version of IPP. Operator and administrator requirements are out of scope for v1.0. The model and semantics document describes a simplified model with abstract objects, their attributes, and their operations. The model introduces a Printer object and a Job object. The Job object supports multiple documents per job. The security document covers potential threats and proposed counters to those threats. The protocol specification is formal document which incorporates the ideas in all the other documents into a concrete mapping using clearly defined data representations and transport protocol mappings that real implementers can use to develop interoperable client and server side components. Finally, the directory schema document shows a generic schema for directory service entries that represent instances of IPP Printers.

This document is the "Internet Printing Protocol/1.0: Protocol Specification" document.

46 Table of Contents

47	1. Introduction	3
48	2. Conformance Terminology.....	3
49	3. Encoding of the Operation Layer.....	3
50	3.1 Picture of the Encoding.....	3
51	3.2 Syntax of Encoding.....	5
52	3.3 Version	6
53	3.4 Mapping of Operations.....	6
54	3.5 Mapping of Status-code.....	7
55	3.6 Tags	8
56	3.7 Name-Lengths.....	9
57	3.8 Mapping of Parameter Names	10
58	3.9 Value Lengths.....	10
59	3.10 Mapping of Attribute and Parameter Values.....	11
60	3.11 Data	12
61	4. Encoding of Transport Layer.....	12
62	4.1 General Headers.....	13
63	4.2 Request Headers.....	13
64	4.3 Response Headers.....	14
65	4.4 Entity Headers.....	14
66	5. Security Considerations.....	14
67	6. Appendix A: Protocol Examples.....	17
68	6.1 Print-Job Request	17
69	6.2 Print-Job Response (successful).....	17
70	6.3 Print-Job Response (failure)	18
71	6.4 Print-URI Request.....	18
72	6.5 Create-Job Request.....	19
73	6.6 Get-Jobs Request.....	19
74	6.7 Get-Jobs Response.....	19
75	7. Appendix B: Requirements without HTTP/1.1 as a Transport Layer.....	20
76	7.1 Additional Parameter-group for HTTP header information.....	20
77	7.2 Chunking of Data.....	21
78	7.3 Revised Picture for the Operation Layer	22
79	7.4 Revised Syntax for the Operation Layer.....	22
80	8. Appendix C: Mapping of Each Operation in the Encoding.....	23
81	9. References.....	15
82	10. Author's Address	16
83	11. Other Participants:	16
84		
85		
86		

87 1. Introduction

88 This document contains the rules for encoding IPP operations and describes two layers: the transport layer and the operation
89 layer.

90 The transport layer consists of an HTTP/1.1 request or response. RFC 2068 [27] describes HTTP/1.1. This document specifies
91 the HTTP headers that an IPP implementation supports.

92 The operation layer consists of a message body in an HTTP request or response. The document "Internet Printing
93 Protocol/1.0: Model and Semantics" [21] defines the semantics of such a message body and the supported values. This
94 document specifies the encoding of an IPP operation. The aforementioned document [21] is henceforth referred to as the "IPP
95 model document"

96 2. Conformance Terminology

97 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
98 "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [25].

99 3. Encoding of the Operation Layer

100 The operation layer SHALL contain a single operation request or operation response.

101 The encoding consists of octet as the most primitive type. There are several types built from octets, but two important primitive
102 types are integers and characters, on which most all other data types are built. Every character in this encoding SHALL be a
103 member of the UCS-2 coded character set and SHALL be encoded using UTF-8 which uses 1 to 3 octets per character. Every
104 integer in this encoding SHALL be encoded in binary as a signed integer using two's-complement binary encoding with big-
105 endian format (also known as "network order" and "most significant byte first"). The number of octetsbytes for an integer
106 SHALL a power of 2, that is, be 1, 2 or 4, depending on usage in the protocol. Such one-octet integers, henceforth called
107 SIGNED-BYTE, are used for the version and tag fields. Such two-byte integers, henceforth called SIGNED-SHORT are used
108 for the operation, status-code and length fields. Four byte integers, henceforth called SIGNED-INTEGER, are used for values
109 fields.

110 The following two sections present the operation layer in two ways

- 111 • informally through pictures and description
- 112 • formally through Augmented Backus-Naur Form (ABNF), as specified by draft-ietf-drums-abnf-02.txt [29]

113 3.1 Picture of the Encoding

114 The encoding for an operation request or response consists of:

115	-----		
116		version	2 bytes - required
117	-----		
118		operation (request) or status-code (response)	2 bytes - required
119	-----		
120		parameter-tag	1 byte
121	-----		- optional
122		parameter-sequence	m bytes
123	-----		
124		attribute-tag	1 byte
125	-----		- 0 or more
126		attribute-sequence	n bytes
127	-----		
128		data-tag	1 byte - required
129	-----		
130		data	q bytes - optional
131	-----		

132 The parameter-tag and parameter-sequence may be omitted if the operation has no parameters. The attribute-tag and attribute-
 133 sequence may be omitted if the operation has no attributes or it may be replicated for an operation that contains attributes for
 134 multiple objects. The data is omitted from some operations, but the data-tag is present even when the data is omitted. Note, the
 135 parameter-tag, attribute-tag and data-tag are called 'delimiter-tags'.

136 A parameter-sequence consists of a sequence of zero or more compound-parameters.

137	-----		
138		compound-parameter	r bytes - 0 or more
139	-----		

140 An attributes-sequence consists of zero or more compound-attributes.

141	-----		
142		compound-attribute	s bytes - 0 or more
143	-----		

144 A compound-parameter consists of a parameter with a single value optionally followed by zero or more additional values. A
 145 compound-attribute consists an attribute with a single value followed by zero or more additional values.

146 Each parameter or attribute consists of:

147	-----		
148		value-tag	1 byte
149	-----		
150		name-length (value is u)	2 bytes
151	-----		
152		name	u bytes
153	-----		
154		value-length (value is v)	2 bytes
155	-----		
156		value	v bytes
157	-----		

158 An additional value consists of:

159	-----			
160		value-tag		1 byte
161	-----			
162		name-length (value is 0x0000)		2 bytes
163	-----			
164		value-length (value is w)		2 bytes
165	-----			
166		value		w bytes
167	-----			
168				

- 0 or more

169 Note: an additional value is like a parameter or attribute whose name-length is 0.

170 From the standpoint of a parsing loop, the encoding consists of:

171	-----			
172		version		2 bytes - required
173	-----			
174		operation (request) or status-code (response)		2 bytes - required
175	-----			
176		tag (delimiter-tag or value-tag)		1 byte
177	-----			
178		empty or rest of parameter/attribute		x bytes
179	-----			
180		data-tag		2 bytes - required
181	-----			
182		data		y bytes - optional
183	-----			
184				

- 0 or more

185 The value of the tag determines whether the bytes following the tag are:

- 186 • parameters
- 187 • attributes
- 188 • data
- 189 • the remainder of a single parameter or attribute where the tag specifies the type of the value.

190 3.2 Syntax of Encoding

191 The syntax below is ABNF except 'strings of literals' SHALL be case sensitive. For example 'a' means lower case 'a' and not
 192 upper case 'A'. In addition, SIGNED-BYTE and SIGNED-SHORT ~~two-byte binary signed integer~~ fields are represented as
 193 '%x' values which show their range of values.

```

194 ipp-message = ipp-request / ipp-response
195 ipp-request = version operation [parameter-tag parameter-sequence ]
196             *(attribute-tag attribute-sequence) data-tag data
197 ipp-response = version status-code [parameter-tag parameter-sequence ]
198             *(attribute-tag attribute-sequence) data-tag data
199
200 version = major-version minor-version
201 major-version = SIGNED-BYTE ; initially %d1
202 minor-version = SIGNED-BYTE ; initially %d0
203
204 operation = SIGNED-SHORT ; mapping from model defined below

```

205 status-code = SIGNED-SHORT ; mapping from model defined below
 206
 207 parameter-sequence = *compound-parameter
 208 attribute-sequence = *compound-attribute
 209 compound-parameter = parameter *additional-values
 210 compound-attribute = attribute *additional-values
 211
 212 parameter = value-tag name-length name value-length value
 213 attribute = value-tag name-length name value-length value
 214 additional-values = value-tag zero-name-length value-length value
 215
 216 name-length = SIGNED-SHORT ; number of octets of 'name'
 217 name = LALPHA *(LALPHA / DIGIT / "-" / "_" / ".")
 218 value-length = SIGNED-SHORT ; number of octets of 'value'
 219 value = OCTET-STRING
 220
 221 data = OCTET-STRING
 222
 223 zero-name-length = %x00.00 ; name-length of 0
 224 parameter-tag = %x01 ; tag of 1
 225 attribute-tag = %x02 ; tag of 2
 226 data-tag = %x03 ; tag of 3
 227 value-tag = %x10-~~..%xFF~~
 228
 229 SIGNED-BYTE = ~~BYTE%x00..%xFF~~
 230 SIGNED-SHORT = ~~2BYTE%x00..%xFF %x00..%xFF~~
 231 DIGIT = ~~%x30-39 ; "0" to "9"~~
 232 LALPHA = ~~%x61-7A ; "a" to "z"~~
 233 BYTE = ~~%x00-..%xFF~~
 234 OCTET-STRING = *BYTE
 235

236 The syntax allows a parameter-tag to be present when the parameter-sequence that follows is empty. The same is true for the
 237 attribute-tag and the attribute-sequence that follows. The syntax is defined this way to allow for the response of Get-Jobs where
 238 no attributes are returned for some job-objects. Although it is RECOMMENDED that the sender not send a parameter-tag if
 239 there are no parameters and not send an attribute-tag if there are no attributes (except in the Get-Jobs response just mentioned),
 240 the receiver MUST be able to decode such syntax.

241 3.3 Version

242 The version SHALL consist of a major and minor version, each of which SHALL be represented by a SIGNED-BYTE~~one-byte~~
 243 ~~signed integer~~. The protocol described in this document SHALL have a major version of 1 (0x01) and a minor version of 0
 244 (0x00). The ABNF for these two bytes SHALL be %x01.00.

245 3.4 Mapping of Operations

246 The following SHALL be the mapping of operations names to integer values which are encoded as a SIGNED-SHORT~~two-byte~~
 247 ~~binary signed integers~~. The operations are defined in the IPP model document. The table below includes a range of values for
 248 future extensions to the protocol and a separate range for private extensions. It is RECOMMENDED that the private extension
 249 values be used for temporary experimental implementations and not for deployed products.

Encoding (hex)	Operation
0x0	reserved (not used)
0x1	Get-Operations
0x2	Print-Job
0x3	Print-URI
0x4	Validate-Job
0x5	Create-Job
0x6	Send-Document
0x7	Send-URI
0x8	Cancel-Job
0x9	Get-Attributes
0xA	Get-Jobs
0xB-0x3FFF	reserved for future operations
0x4000-0xFFFF	reserved for private extensions

250 3.5 Mapping of Status-code

251 The following SHALL be the mapping of status-code names to integer values which are encoded as a SIGNED-SHORT-two
 252 byte binary signed integers. The status-code names are defined in the IPP model document.

253 If an IPP status-code is returned, then the HTTP Status-Code MUST be 200 (OK). With any other HTTP Status-Code value, the
 254 HTTP response SHALL NOT contain an IPP message-body, and thus no IPP status-code is returned.

255 Note: the integer encodings below were chosen to be similar to corresponding Status-Code values in HTTP. The IPP client and
 256 server errors have the same relative offset to their base as corresponding HTTP errors, but the IPP base is a multiple of 0x100
 257 whereas the HTTP base is a multiple of 100. Despite this similarity, the Status-Code returned at the HTTP level will always be
 258 different except in the case where 'OK' is returned at both levels, 200 (OK) in HTTP and 0 (successful-OK) in IPP.

259 Note: some status-code values, such as client-error-unauthorized, may be returned at the transport (HTTP) level rather than the
 260 operation level.

261 ISSUE: as implementations proceed, we will learn what error code need to be supported at the IPP level.

Encoding (hex)	Status-Code Name
0	successful-OK
0x400	client-error-bad-request
0x401	client-error-unauthorized
0x403	client-error-forbidden
0x404	client-error-not-found
0x405	client-error-method-not-allowed
0x408	client-error-timeout
0x40A	client-error-gone
0x40D	client-error-request-entity-too-large
0x40E	client-error-request-URI-too-long
0x40F	client-error-unsupported-document-format
0x410	client-error-attribute-not-supported
0x500	server-error-internal-server-error
0x501	server-error-operation-not-implemented
0x503	server-error-service-unavailable
0x504	server-error-timeout
0x505	server-error-version-not-supported

Encoding (hex)	Status-Code Name
0x506	server-error-printer-error
0x507	server-error-temporary-error

262 3.6 Tags

263 There are two kinds of tags:

- 264 • delimiter tags: delimit major sections of the protocol, namely parameters, attributes and data
- 265 • value tags: specify the type of each parameter or attribute value

266 3.6.1 Delimiter Tags

267 The following table specifies the values for the delimiter tags:

Tag Value (Hex)	Delimiter
0x00	reserved
0x01	parameter-tag
0x02	attribute-tag
0x03	data-tag
0x04-0x0F	reserved for future delimiters

268 3.6.2 Value Tags

269 The remaining tables show values for the value-tag, which is the first octet of a parameter or attribute. The value-tag specifies
 270 the type of the value of the parameter or attribute. The value of the value-tag of a parameter or attribute SHALL either be a type
 271 value specified in the model document or an “out-of-band” value, such as “unsupported” or “default”. If the value of value-tag
 272 for a attribute or parameter is not “out-of-band” and differs from the value type specified by the model document, then a server
 273 receiving such a request MAY reject it, and a client receiving such a response MAY ignore the attribute or parameter.

274 The following table specifies the “out-of-band” values for the value-tag.

Tag Value (Hex)	Meaning
0x10	unsupported
0x11	default
<u>0x12</u>	<u>no-value</u>
0x13-0x1F	reserved for future “out-of-band” values.

275 The “unsupported” value SHALL be used in the attribute-sequence of an error response for those attributes which the server
 276 does not support. The “default” value is reserved for future use of setting value back to their default value. The “no-value” value
 277 is used for the “no-value” value in model, e.g. when a document-attribute is returned as a set of values and an attribute has no
 278 specified value for one or more of the documents.

279 The following table specifies the integer values for the value-tag

Tag Value (Hex)	Meaning
0x20	reserved

Tag Value (Hex)	Meaning
0x21	integer (up to 4 bytes)
0x22	boolean
0x23	enum seconds
0x24	milli seconds
0x25	enum
0x24-0x23F	reserved for future integer types

280 NOTE: 0x20 is reserved for “generic integer” if should ever be needed.

281 The following table specifies the octet-string values for the value-tag

<u>Tag Value (Hex)</u>	<u>Meaning</u>
<u>0x30</u>	<u>reserved</u>
<u>0x31</u>	<u>dateTime</u>
<u>0x32</u>	<u>resolution</u>
<u>0x33-0x3F</u>	<u>reserved for future octet-string types</u>

282 The following table specifies the character-string values for the value-tag

Tag Value (Hex)	Meaning
0x40	reserved
0x41	text
0x42	name
0x43	language filename
0x44	keyword
0x45	uri
0x46	uriScheme
0x47	dateTime
0x47-0x57F	reserved for future character string types

283 NOTE: 0x40 is reserved for “generic character-string” if should ever be needed.

284 The values 0x680-0xFF are reserved for future types. There are no values allocated for private extensions. A new type must be
 285 registered via the type 2 process.

286 Issue: should this be a type 1 process.

287 3.7 Name-Lengths

288 The name-length field SHALL consist of a ~~SIGNED-SHORT~~~~two-byte binary signed integer in big-endian order~~. This field
 289 SHALL specify the number of octets in the name field which follows the name-length field, excluding the two bytes of the
 290 name-length field.

291 If a name-length field has a value of zero, the following name field SHALL be empty, and the following value SHALL be
 292 treated as an additional value for the preceding parameter. Within a parameter-sequence, if two parameters have the same
 293 name, the first occurrence SHALL be ignored. Within an attribute-sequence, if two attributes have the same name, the first
 294 occurrence SHALL be ignored. The zero-length name is the only mechanism for multi-valued parameters and attributes.

295 3.8 Mapping of Parameter Names

296 Some parameters are encoded in a special position. These parameters are:

- 297 • ~~“printer-uri”: The printer-uri of each printer object operation in the IPP model document SHALL be specified both as~~
298 ~~a parameter named “printer-uri” in the operation layer ,and outside of the operation layer as the request-URI on the~~
299 ~~Request-Line at the HTTP level.,~~
- 300 • ~~“job-uri”: The job -uri of each job object operation in the IPP model document SHALL be specified both as a~~
301 ~~parameter named “job -uri” in the operation layer ,and outside of the operation layer as the request-URI on the~~
302 ~~Request-Line at the HTTP level.,~~
- 303 • ~~“request-URI”: The request-URI of each operation in the IPP model document SHALL be specified outside of the~~
304 ~~operation layer as the request-URI on the Request-Line at the HTTP level, and SHALL not be specified within the~~
305 ~~operation layer.~~
- 306 • “document-content”: The parameter named “document-content” in the IPP model document SHALL become the
307 “data” in the operation layer.
- 308 • “status-code”: The parameter named “status-code” in the IPP model document SHALL become the “status-code” field
309 in the operation layer response.

310 ~~ISSUE: Should the request-URI that is the target object of the operation be outside the operation layer, or should it be inside as~~
311 ~~a parameter and a separate print server URI outside in the HTTP Request-Line?~~

312 The remaining parameters are encoded in the parameter-sequence or the attribute-sequence. The parameter-sequence is for
313 actual operation parameters and the attribute-sequence is for object attributes. Of the parameters defined in the IPP model
314 document, some represent an actual operation parameters and others represent a collection of object attributes.

315 A parameter in the IPP model document SHALL represent a collection of object attributes if its name contains the word
316 “attributes” immediately preceded by a space; otherwise it SHALL represent an actual operation parameter. Note, some actual
317 operation parameters contain the word “attributes” preceded by a hyphen (“-”). They are not a collection of attributes.

318 If a parameter in IPP model document represents an actual operation parameter and is not in a special position, it SHALL be
319 encoded in the parameter-sequence using the text name of the parameter specified in the IPP model document.

320 If a parameter in IPP model document represents a collection of object attributes, the attributes SHALL be encoded in the
321 attribute-sequence using the text names of the attributes specified in the IPP model document. The IPP model document
322 specifies the members of such attribute collections, but does not require that all members of a collection be present in an
323 operation.

324 If an operation contain attributes from exactly one object, there SHALL be exactly one attribute-sequence. If an operation
325 contains attributes from more than one object (e.g. Get-Jobs response), the attributes from each object SHALL be in a separate
326 attribute-sequence, such that the attributes from the ith object are in the ith attribute-sequence.

327 See Section 10 “Appendix **BC**: Mapping of Each Operation in the Encoding” for table showing the application of the rules
328 above.

329 3.9 Value Lengths

330 Each parameter value SHALL be preceded by a ~~SIGNED-SHORT~~~~two-byte binary signed integer in big-endian order~~ which
331 SHALL specify the number of octets in the value which follows this length, exclusive of the two bytes specifying the length.

332 For any of the types represented by binary signed integers, the sender ~~MUST~~~~MAY~~ encode the value in ~~exactly four octets..fewer~~
333 ~~than the maximum 4 bytes, but the number of bytes for the encoding MUST be a power of two, i.e. 1, 2 or 4, and representation~~
334 ~~MUST be as a signed integer in the fewer bytes.~~

335 ~~ISSUE: allowing 4 byte integers to be transmitted in 1 or 2 bytes, at client discretion, may be more trouble than the saved bytes~~
 336 ~~are worth. Do we really want this?~~

337 For any of the types represented by character-strings, the sender MUST encode the value with all the characters of the string
 338 and without any padding characters.

339 If a value-tag contains an “out-of-band” value, such as “unsupported”, the value-length SHALL be 0 and the value empty —
 340 the value has no meaning when the value-tag has an “out-of-band” value. If a server or client receives an operation with a
 341 nonzero value-length in this case, it SHALL ignore the value field.

342 3.10 Mapping of Attribute and Parameter Values

343 The following SHALL be the mapping of attribute and parameter values to their IPP encoding in the value field. The syntax
 344 types are defined in the IPP model document.

Syntax of Attribute Value	Encoding
text	an octet string where each character is a member of the UCS-2 coded character set and is encoded using UTF-8. The text is encoded in “network byte order” with the first character in the text (according to reading order) being the first character in the encoding.
name	same as text
fileName	same as text
language	same as text but with a syntax specified by RFC 1766
keyword	same as text. Allowed text values are defined in the IPP model document
uri	same as text
uriScheme	same as text
boolean	one binary octet where 0x00 is ‘false’ and 0x01 is ‘true’
integer	number of octets is a power of 2 (i.e. 1, 2 or 4). These a SIGNED-INTEGERS, defined previously as a signed integer using two’s-complement binary encoding in four octets with big-endian format (also known as “network order” and “most significant byte first”). octets represent a binary signed integer in big-endian order (also known as “network byte order” and MSB first).
enum	same as integer. Allowed integer values are defined in the IPP model document
dateTime	eleven octets whose contents are defined by “DateAndTime” in RFC 1903. Although RFC 1903 also defines an eight octet format which omits the time zone, a value of this type in the IPP protocol MUST use the eleven octet format. same as text. Syntax of data and time is defined by ISO 8601 ISSUE: should ISO 8601 be called out in the IPP model document?
resolution	nine octets consisting of 2 SIGNED-INTEGERS followed by a SIGNED-BYTE. The values are the same as those specified in draft-ietf-printmib-mib-info-02.txt [30]. The first SIGNED-INTEGERS contains the value of prtMarkerAddressabilityXFeedDir. The second SIGNED-INTEGERS contains the value of prtMarkerAddressabilityFeedDir. The SIGNED-BYTE contains the value of prtMarkerAddressabilityUnit. Note: the latter value is either 3 (tenThousandsOfInches) or 4 (micrometers) and the addressability is in 10,000 units of measure. Thus the SIGNED-INTEGERS represent integral values in either dots-per-inch or dots-per-centimeter.
seconds	same as integer
milliseconds	same as integer
1setOf X	encoding according to the rules for a parameter with more than more value. Each value X is encoded according to the rules for encoding its type.
rangeOf X	same 1setOf X where the number of values is 2.

345 The type of the value in the model document determines the encoding in the value and the value of the value-tag.

346 **3.11 Data**

347 The data part SHALL include any data required by the operation

348 **4. Encoding of Transport Layer**

349 HTTP/1.1 shall be the transport layer for this protocol.

350 The operation layer has been designed with the assumption that the transport layer contains the following information:

- 351 • the URI of the target job or printer operation
- 352 • the total length of the data in the operation layer, either as a single length or as a sequence of chunks each with a
- 353 length.
- 354 • the client's language, the character-set and the transport encoding.

355 Each HTTP operation shall use the POST method where the request-URI is the object target of the operation, and where the
356 "Content-Type" of the message-body in each request and response shall be "application/ipp". The message-body shall contain
357 the operation layer and shall have the syntax described in section 3.2 "Syntax of Encoding".

358 A client implementation SHALL adhere to the rules for a client described in RFC 2068. A server implementation SHALL
359 adhere the rules for an origin server described in RFC 2068.

360 In the following sections, there are a tables of all HTTP headers which describe their use in an IPP client or server. The
361 following is an explanation of each column in these tables.

- 362 • the "header" column contains the name of a header
- 363 • the "request/client" column indicates whether a client sends the header.
- 364 • the "request/server" column indicates whether a server supports the header when received.
- 365 • the "response/server" column indicates whether a server sends the header.
- 366 • the "response /client" column indicates whether a client supports the header when received.
- 367 • the "values and conditions" column specifies the allowed header values and the conditions for the header to be present
- 368 in a request/response.

369 The table for "request headers" does not have columns for responses, and the table for "response headers" does not have
370 columns for requests.

371 The following is an explanation of the values in the "request/client" and "response/server" columns.

- 372 • **must:** the client or server MUST send the header,
- 373 • **must-if:** the client or server MUST send the header when the condition described in the "values and conditions"
374 column is met,
- 375 • **may:** the client or server MAY send the header
- 376 • **not:** the client or server SHOULD NOT send the header. It is not relevant to an IPP implementation.

377 The following is an explanation of the values in the "response/client" and "request/server" columns.

- 378 • **must:** the client or server MUST support the header,
- 379 • **may:** the client or server MAY support the header

- 380 • **not:** the client or server SHOULD NOT support the header. It is not relevant to an IPP implementation.

381 4.1 General Headers

382 The following is a table for the general headers.

383 ISSUE: an HTTP expert should review these tables for accuracy.

General-Header	Request		Response		Values and Conditions
	Client	Server	Server	Client	
Cache-Control Connection	must must-if	not must	must must-if	not must	“no-cache” only “close” only. Both client and server SHOULD keep a connection for the duration of a sequence of operations. The client and server MUST include this header for the last operation in such a sequence.
Date	may	may	must	may	per RFC 1123 [9]
Pragma`	must	not	must	not	“no-cache” only
Transfer-Encoding	must-if	must	must-if	must	“chunked” only . Header MUST be present if Content-Length is absent.
Upgrade	not	not	not	not	
Via	not	not	not	not	

384

385 4.2 Request Headers

386 The following is a table for the request headers.

387

Request-Header	Client	Server	Request Values and Conditions
Accept	may	must	“application/ipp” only. This value is the default if the client omits it per IANA Character Set registry. ISSUE: is this useful for IPP? empty and per RFC 2068 [27] and IANA registry for content-codings see RFC 1766 [26]. A server SHOULD honor language requested by returning the values status-message, job-state-message and printer-state-reason in one of requested languages.
Accept-Charset	may	must	
Accept-Encoding	may	must	
Accept-Language	may	must	
Authorization	must-if	must	per RFC 2068. A client MUST send this header when it receives a 401 “Unauthorized” response and does not receive a “Proxy-Authenticate” header.
From	not	not	per RFC 2068. Because RFC recommends sending this header only with the user’s approval, it is not very useful
Host	must	must	per RFC 2068
If-Match	not	not	
If-Modified-Since	not	not	
If-None-Match	not	not	
If-Range	not	not	
If-Unmodified-Since	not	not	
Max-Forwards	not	not	
Proxy-Authorization	must-if	not	per RFC 2068. A client MUST send this header when it receives a 401 “Unauthorized” response and a “Proxy-Authenticate” header.
Range	not	not	
Referer	not	not	

Request-Header	Client	Server	Request Values and Conditions
User-Agent	not	not	

388 4.3 Response Headers

389 The following is a table for the request headers.
390

Response-Header	Server	Client	Response Values and Conditions
Accept-Ranges	not	not	
Age	not	not	
Location	must-if	may	per RFC 2068. When URI needs redirection.
Proxy-Authenticate	not	must	per RFC 2068
Public	may	may	per RFC 2068
Retry-After	may	may	per RFC 2068
Server	not	not	
Vary	not	not	
Warning	may	may	per RFC 2068
WWW-Authenticate	must-if	must	per RFC 2068. When a server needs to authenticate a client.

391 4.4 Entity Headers

392 The following is a table for the entity headers.
393

Entity-Header	Request		Response		Values and Conditions
	Client	Server	Server	Client	
Allow	not	not	not	not	
Content-Base	not	not	not	not	
Content-Encoding	may	must	must	must	per RFC 2068 and IANA registry for content codings.
Content-Language	may	must	must	must	see RFC 1766 [26].
Content-Length	must-if	must	must-if	must	the length of the message-body per RFC 2068. Header MUST be present if Transfer-Encoding is absent..
Content-Location	not	not	not	not	
Content-MD5	may	may	may	may	per RFC 2068
Content-Range	not	not	not	not	
Content-Type	must	must	must	must	“application/ipp” only
ETag	not	not	not	not	
Expires	not	not	not	not	
Last-Modified	not	not	not	not	

394 5. Security Considerations

395 When utilizing HTTP 1.1 as a transport of IPP, the security considerations outlined in RFC 2068 apply. Specifically, IPP
396 servers can generate a 401 "Unauthorized" response code to request client authentication and IPP clients should correctly
397 respond with the proper "Authorization" header. Both Basic Authentication (RFC 2068) and Digest Authentication (RFC
398 2069) flavors of authentication should be supported. The server chooses which type(s) of authentication to accept. Digest
399 Authentication is a more secure method, and is always preferred to Basic Authentication.

400 For secure communication (privacy in particular), IPP should be run using a secure communications channel. Both Transport
401 Layer Security - TLS (draft-ietf-tls-protocol-03) and IPSec (RFC 1825) provide necessary features for security. It is possible to

402 combine the two techniques, HTTP 1.1 client authentication (either basic or digest) with secure communications channel
403 (either TLS or IPSec). Together the two are more secure than client authentication and they perform user authentication.

404 Complete discussion of IPP security considerations can be found in the IPP Security document

405 [ISSUE: how does each security mechanism supply the job-originating-user and job-originating-host values?](#)

406 6. References

- 407 [1] Smith, R., Wright, F., Hastings, T., Zilles, S., and Gyllenskog, J., "Printer MIB", RFC 1759, March 1995.
- 408 [2] Berners-Lee, T, Fielding, R., and Nielsen, H., "Hypertext Transfer Protocol - HTTP/1.0", RFC 1945, August 1995.
- 409 [3] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", RFC 822, August 1982.
- 410 [4] Postel, J., "Instructions to RFC Authors", RFC 1543, October 1993.
- 411 [5] ISO/IEC 10175 Document Printing Application (DPA), June 1996.
- 412 [6] Herriot, R. (editor), X/Open A Printing System Interoperability Specification (PSIS), August 1995.
- 413 [7] Kirk, M. (editor), POSIX System Administration - Part 4: Printing Interfaces, POSIX 1387.4 D8, 1994.
- 414 [8] Borenstein, N., and Freed, N., "MIME (Multi-purpose Internet Mail Extensions) Part One: Mechanism for Specifying
415 and Describing the Format of Internet Message Bodies", RFC 1521, September, 1993.
- 416 [9] Braden, S., "Requirements for Internet Hosts - Application and Support", RFC 1123, October, 1989,
- 417 [10] McLaughlin, L. III, (editor), "Line Printer Daemon Protocol" RFC 1179, August 1990.
- 418 [11] Berners-Lee, T., Masinter, L., McCahill, M. , "Uniform Resource Locators (URL)", RFC 1738, December, 1994.
- 419 [20] Wright, F. D., "[Requirements for an](#) Internet Printing Protocol: ~~Requirements~~"
- 420 [21] Isaacson, S, deBry, R, Hasting, T, Herriot, R, Powell, P, "Internet Printing Protocol/1.0: Model and Semantics"
- 421 [22] Internet Printing Protocol/1.0: Security
- 422 [23] Herriot, R, Butler, S, Moore, P, Turner, R, "Internet Printing Protocol/1.0: Protocol Specification" (This document)
- 423 [24] Carter, K, Isaacson, S, "Internet Printing Protocol/1.0: Directory Schema"
- 424 [25] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119 , March 1997
- 425 [26] H. Alvestrand, " Tags for the Identification of Languages", RFC 1766, March 1995.
- 426 [27] R Fielding, et al, "Hypertext Transfer Protocol – HTTP/1.1" RFC 2068, January 1997
- 427 [28] [J. Case, et al. "Textual Conventions for Version 2 of the Simple Network Management Protocol \(SNMPv2\)", RFC](#)
428 [1903, January 1996. Marcus Kuhn, "International Standard Date and Time Notation", ISO 8601,](#)
429 <http://www.ft.uni-erlangen.de/~mskuhn/iso-time.html>

430 [29] D. Crocker et al., "Augmented BNF for Syntax Specifications: ABNF", draft-ietf-drums-abnf-032.txt.

431 [\[30\] R. Turner, "Printer MIB", draft-ietf-printmib-mib-info-02.txt, July 12, 1997.](#)

432 7. Author's Address

433

Robert Herriot (editor)
Sun Microsystems Inc.
901 San Antonio Road, MPK-17
Palo Alto, CA 94303

Phone: ~~650415~~-786-8995
Fax: ~~650415~~-786-7077
Email: robert.herriot@eng.sun.com

Sylvan Butler
Hewlett-Packard
11311 Chinden Blvd.
Boise, ID 83714

Phone: 208-396-6000
Fax: 208-396-3457
Email: sbutler@boi.hp.com

IPP Mailing List: ipp@pwg.org
IPP Mailing List Subscription: ipp-request@pwg.org
IPP Web Page: <http://www.pwg.org/ipp/>

434

435 8. Other Participants:

Chuck Adams - Tektronix
Ron Bergman - Data Products
Keith Carter - IBM
Angelo Caruso - Xerox
Jeff Copeland - QMS
Roger Debry - IBM
Lee Farrell - Canon
Sue Gleeson - Digital
Charles Gordon - Osicom
Brian Grimshaw - Apple
Jerry Hadsell - IBM
Richard Hart - Digital
Tom Hastings - Xerox
Stephen Holmstead
Zhi-Hong Huang - Zenographics
Scott Isaacson - Novell
Rich Lomicka - Digital
David Kellerman - Northlake Software

Paul Moore
Microsoft
One Microsoft Way
Redmond, WA 98053

Phone: 425-936-0908
Fax: 425-93MS-FAX
Email: paulmo@microsoft.com

Randy Turner
Sharp Laboratories
5750 NW Pacific Rim Blvd
Camas, WA 98607

Phone: 360-817-8456
Fax: : 360-817-8436
Email: rturner@sharplabs.com

Harry Lewis - IBM
Tony Liao - Vivid Image
David Manchala - Xerox
Carl-Uno Manros - Xerox
Jay Martin - Underscore
Larry Masinter - Xerox
Bob Pentecost - Hewlett-Packard
Patrick Powell - SDSU
Jeff Rackowitz - Intermec
Xavier Riley - Xerox
Gary Roberts - Ricoh
Stuart Rowley - Kyocera
Richard Schneider - Epson
Shigern Ueda - Canon
Bob Von Anandel - Allegro Software
William Wagner - Digital Products
Jasper Wong - Xionics
Don Wright - Lexmark

Robert Kline - TrueSpectra
 Dave Kuntz - Hewlett-Packard
 Takami Kurono - Brother
 Rich Landau - Digital
 Greg LeClair - Epson

Rick Yardumian - Xerox
 Lloyd Young - Lexmark
 Peter Zehler - Xerox
 Frank Zhao - Panasonic
 Steve Zilles - Adobe

436 9. Appendix A: Protocol Examples

437 9.1 Print-Job Request

438 The following is an example of a Print-Job request with job-name, copies, and sides specified.

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0002	PrintJob	operation
0x02	start attributes	attribute tag
0x42	name type	value-tag
0x0008		name-length
job-name	job-name	name
0x0006		value-length
foobar	foobar	value
0x21	integer type	value-tag
0x0005		name-length
copies	copies	name
0x0004		value-length
0x000014	20	value
0x44	keyword type	value-tag
0x0005		name-length
sides	sides	name
0x0001		value-length
two-sided-long-edge	two-sided-long-edge	value
0x03	start-data	data-tag
%!PS...	<PostScript>	data

439 9.2 Print-Job Response (successful)

440 Here is an example of a Print-Job response which is successful:

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0000	OK (successful)	status-code
0x01	start parameters	parameter tag
0x41	text type	value-tag
0x000E		name-length
status-message	status-message	name
0x0002		value-length
OK	OK	value
0x45	uri type	value-tag
0x0008		name-length
job-uri	job-uri	name

Octets	Symbolic Value	Protocol field
0x000E		value-length
http://foo/123	http://foo/123	value
0x02	start attributes	attribute tag
0x25	name type	value-tag
0x0008		name-length
job-state	job-state	name
0x0001		value-length
0x03	pending	value
0x03	start-data	data-tag

441 9.3 Print-Job Response (failure)

442 Here is an example of a Print-Job response which fails because the printer does not support sides and because the value 20 for
443 copies is not supported:

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0400	client-error-bad-request	status-code
0x01	start parameters	parameter tag
0x41	text type	value-tag
0x000E		name-length
status-message	status-message	name
0x000D		value-length
bad-request	bad-request	value
0x02	start attributes	attribute tag
0x21	integer type	value-tag
0x0005		name-length
copies	copies	name
0x0004		value-length
0x000014	20	value
0x10	unsupported (type)	value-tag
0x0005		name-length
sides	sides	name
0x0000		value-length
0x03	start-data	data-tag

444 9.4 Print-URI Request

445 The following is an example of Print-URI request with copies and job-name parameters.

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0003	Print-URI	operation
0x01	start-parameters	parameter tag
0x45	uri type	value-tag
0x000A		name-length
document-uri	document-uri	name
0x11		value-length
ftp://foo.com/foo	ftp://foo.com/foo	value
0x02	start-attributes	attribute tag

Octets	Symbolic Value	Protocol field
0x42	name type	value-tag
0x0008		name-length
job-name	job-name	name
0x0006		value-length
foobar	foobar	value
0x21	integer type	value-tag
0x0005		name-length
copies	copies	name
0x0004+		value-length
0x000001	1	value
0x03	start-data	data-tag
%!PS...	<PostScript>	data

446 9.5 Create-Job Request

447 The following is an example of Create-Job request with no parameters and no attributes

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0005	Create-Job	operation
0x03	start-data	data-tag

448 9.6 Get-Jobs Request

449 The following is an example of Get-Jobs request with parameters but no attributes.

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x000A	Get-Jobs	operation
0x01	start-parameters	parameter-tag
0x21	integer type	value-tag
0x0005		name-length
limit	limit	name
0x0004+		value-length
0x000032	50	value
0x44	keyword type	value-tag
0x0014		name-length
requested-attributes	requested-attributes	name
0x0007		value-length
job-uri	job-uri	value
0x44	keyword type	value-tag
0x0000	additional value	name-length
0x0008		value-length
job-name	job-name	value
0x03	start-data	data-tag

450 9.7 Get-Jobs Response

451 The following is an of Get-Jobs response from previous request with 3 jobs. The Printer returns no information about the
452 second job.

Octets	Symbolic Value	Protocol field
--------	----------------	----------------

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0000	OK (successful)	status-code
0x01	start-parameters	parameter-tag
0x41	text type	value-tag
0x000E		name-length
status-message	status-message	name
0x0002		value-length
OK	OK	value
0x02	start-attributes (1 st object)	attribute-tag
0x45	uri type	value-tag
0x0007		name-length
job-uri	job-uri	name
0x000E		value-length
http://foo/123	http://foo/123	value
0x42	name type	value-tag
0x0008		name-length
job-name	job-name	name
0x0003		name-length
foo	foo	name
0x02	start-attributes (2 nd object)	attribute-tag
0x02	start-attributes (3 rd object)	attribute-tag
0x45	uri type	value-tag
0x0007		name-length
job-uri	job-uri	name
0x000E		value-length
http://foo/124	http://foo/124	value
0x42	name type	value-tag
0x0008		name-length
job-name	job-name	name
0x0003		name-length
bar	bar	name
0x03	start-data	data-tag

453 ~~10. Appendix B: Requirements without HTTP/1.1 as a Transport Layer~~

454 ~~The operation layer defined above assumed certain services would be provided by the HTTP transport layer. Without that layer,~~
 455 ~~information, such as length, request URI and Content Encoding are absent.~~

456 ~~This section specifies the modifications to the operation layer encoding for raw TCP/IP. The following changes would have to~~
 457 ~~made. All of these changes are upward compatible with the encoding defined in section 3 "Encoding of the Operation Layer".~~

458 ~~9.8 Additional Parameter group for HTTP header information~~

459 ~~There is an additional header sequence which is like a parameter sequence. The header sequence SHALL appear the before~~
 460 ~~the parameter sequence, and it SHALL contain the "request URI" along with relevant HTTP header information, including~~
 461 ~~those shown below. This header sequence SHALL be preceded by a header tag to mark that a header sequence follows.~~

462 ~~The following table shows the mapping of HTTP headers to parameters in the operation layer.~~

HTTP header or other concept	IPP header name	Syntax Type of header
---	----------------------------	----------------------------------

HTTP header or other concept	IPP header name	Syntax	Type of header
URI	request-URI	uri	
Connection	Close-Connection	Boolean	
Accept-Charset	Accept-Charset	keyword	
Accept-Encoding	Accept-Encoding	keyword	
Accept-Language	Accept-Language	keyword	
Content-Encoding	Content-Encoding	keyword	
Content-Language	Content-Language	keyword	
charset-parameter	Content-Charset	keyword	
miscellaneous-security	if needed at this level		

463 ~~The first parameter in the header sequence for a request SHALL be the attribute "request-URI" which is the target object of the~~
 464 ~~operation.~~

465 ~~Except for Content-Encoding, the headers SHALL take effect at the beginning of the parameter sequence and apply to the rest~~
 466 ~~of the operation. If a header is Content-Encoding, then the encoding SHALL apply only to the 'full data' or 'data segment's as~~
 467 ~~defined by the syntax below and the resulting decoded data SHALL have the syntax of all octets that follow a header sequence.~~
 468 ~~The syntax in a section below defines the syntax following a header sequence to be:~~

469 ~~[parameter-tag parameter-sequence] *(attribute-tag attribute-sequence) data~~

471 ~~From a decoding point of view, if Content-Encoding is specified, the operation's data is decoded using the algorithm specified~~
 472 ~~by Content-Encoding. The resulting octet stream is parsed as if it were the octets following a header sequence.~~

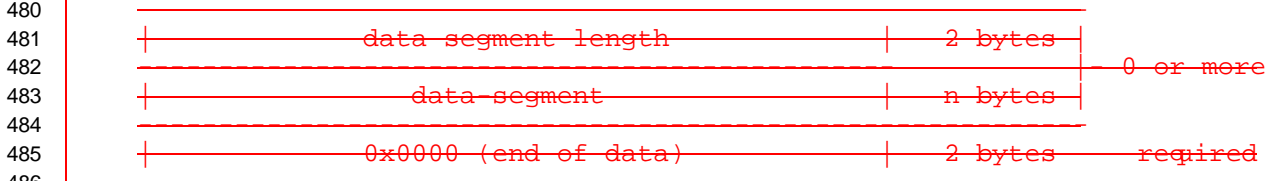
473 ~~NOTE: This rule for Content-Encoding allows a client or server to encode the parameter sequence and attribute sequences or~~
 474 ~~to transmit them un-encoded.~~

475 ~~ISSUE: should the status message be in the header sequence instead of the parameter group for responses?~~

476 ~~9.8 Chunking of Data~~

477 ~~There is a new delimiter tag called 'chunked data tag' which denotes that the following data is chunked. Chunked data consists~~
 478 ~~of a sequence of data segment lengths and data segments. A data segment length of 0 denotes the end of the data.~~

479 ~~The following is the informal picture of chunked data:~~



487 ~~The syntax for the chunked data using ABNF is:~~

488 ~~chunked data = *data chunk end of data~~

489 ~~data chunk = data segment length data segment~~

491 ~~data segment length = SIGNED SHORT ; number of octets of the data in binary~~

492 ~~data segment = OCTET STRING~~

493
494
495
496
497

~~end-of-data = %x00.00~~

~~A data segment contains fragments of the data. When all the data segments are concatenated together, they form the complete data.~~

498 ~~9.8 Revised Picture for the Operation Layer~~

499 ~~The encoding for an operation request or response consists of:~~

500	-----		
501	 -----	version	 2 bytes required
502	-----		
503	 -----	operation (request) or status code (response)	 2 bytes required
504	-----		
505	 -----	header tag	 1 byte
506	-----		
507	 -----	header sequence	 k bytes
508	-----		
509	 -----	parameter tag	 1 byte
510	-----		
511	 -----	parameter sequence	 m bytes
512	-----		
513	 -----	attribute tag	 1 byte
514	-----		
515	 -----	attribute sequence	 n bytes
516	-----		
517	 -----	data tag	 1 byte required
518	-----		
519	 -----	data	 q bytes optional
520	-----		

521 ~~The definition of all fields above is the same as defined in section 3.1 "Picture of the Encoding" except~~

- 522 ~~• 'data' which either has the form defined in the preceding section (9.8 "Revised Picture for the Operation Layer") or~~
- 523 ~~the form described in section 3.1 "Picture of the Encoding".~~
- 524 ~~• 'header-tag' which is new and defined like parameter-tag.~~
- 525 ~~• 'header-sequence' which is new and defined like parameter-sequence.~~

526 ~~9.8 Revised Syntax for the Operation Layer~~

527 ~~The following is the revised syntax for the operation layer.~~

528 ~~ipp-message = ipp-request / ipp-response~~
 529 ~~ipp-request = version operation [header-tag header-sequence]~~
 530 ~~[parameter-tag parameter-sequence] *(attribute-tag attribute-sequence) data~~
 531 ~~ipp-response = version status-code [header-tag header-sequence]~~
 532 ~~[parameter-tag parameter-sequence] *(attribute-tag attribute-sequence) data~~
 533
 534 ~~version = major-version minor-version~~
 535 ~~major-version = SIGNED-BYTE ; initially %d1~~
 536 ~~minor-version = SIGNED-BYTE ; initially %d0~~
 537
 538 ~~operation = SIGNED-SHORT ; mapping from model defined below~~

539 ~~status-code = SIGNED-SHORT ; mapping from model defined below~~
540
541 ~~header-sequence = *compound-header~~
542 ~~parameter-sequence = *compound-parameter~~
543 ~~attribute-sequence = *compound-attribute~~
544
545 ~~compound-header = header *(additional-values)~~
546 ~~compound-parameter = parameter *(additional-values)~~
547 ~~compound-attribute = attribute *(additional-values)~~
548
549 ~~header = value-tag name-length name-value-length value~~
550 ~~parameter = value-tag name-length name-value-length value~~
551 ~~attribute = value-tag name-length name-value-length value~~
552 ~~additional-values = value-tag zero-name-length value-length value~~
553
554 ~~name-length = SIGNED-SHORT ; number of octets of 'name'~~
555 ~~name = LALPHA *(LALPHA / DIGIT / "-" / "_")~~
556 ~~value-length = SIGNED-SHORT ; number of octets of 'value'~~
557 ~~value = OCTET-STRING~~
558
559 ~~data = (data-tag full-data) / (chunked-data-tag chunked-data)~~
560 ~~full-data = OCTET-STRING~~
561 ~~chunked-data = *data-chunk END-OF-DATA~~
562 ~~data-chunk = data-segment-length data-segment~~
563 ~~data-segment-length = SIGNED-SHORT ; number of octets of the data in binary~~
564 ~~data-segment = OCTET-STRING~~
565
566 ~~zero-name-length = %x00.00 ; name-length of 0~~
567 ~~parameter-tag = %x01 ; tag of 1~~
568 ~~attribute-tag = %x02 ; tag of 2~~
569 ~~data-tag = %x03 ; tag of 3~~
570 ~~chunked-data-tag = %x04 ; tag of 4~~
571 ~~header-tag = %x05 ; tag of 5~~
572 ~~value-tag = %x10..%xFF~~
573 ~~end-of-data = %x00.00~~
574
575 ~~SIGNED-BYTE = %x00..%xFF~~
576 ~~SIGNED-SHORT = %x00..%xFF %x00..%xFF~~
577 ~~DIGIT = "0".."9"~~
578 ~~LALPHA = "a".."z"~~
579 ~~BYTE = %x00..%xFF~~
580 ~~OCTET-STRING = *BYTE~~

581 10. Appendix **BC**: Mapping of Each Operation in the Encoding

582 The next three tables show the results of applying the rules above to the operations defined in the IPP model document. There is
583 no information in these tables that cannot be derived from the rules presented in Section 3.8 "Mapping of Parameter Names".

584 The following table shows the mapping of all IPP model document request parameters (~~except request-URI~~) to a parameter-
585 sequence, attribute-sequence or special position in the protocol.

Operation	parameter-sequence	attribute-sequence	special position
-----------	--------------------	--------------------	------------------

Operation	parameter-sequence	attribute-sequence	special position
Get-Operations	printer-uri		
Print-Job	printer-uri best-effort job-name	job-template attributes document attributes	document-content
Print-URI	printer-uri best-effort job-name document-uri	job-template attributes document attributes	
Validate-Job or Create-Job	printer-uri best-effort job-name	job-template attributes	
Send-Document	job-uri last-document	document attributes	document-content
Send-URI	job-uri last-document document-uri	document attributes	
Cancel-Job	job-uri message		
Get-Attributes (for a Printer)	printer-uri document-format requested-attributes		
Get-Attributes (for a Job)	job-uri document-format requested-attributes		
Get-Jobs	printer-uri limit requested-attributes		

586 The following table shows the mapping of all IPP model document response parameters to a parameter-sequence, attribute-
587 sequence or special position in the protocol.

Operation	parameter-sequence	attribute-sequence	special position
Get-Operations	status-message supported-operations		status-code
Print-Job, Print- URI or Create-Job	status-message job-uri	job-status attributes	status-code
Send-Document or Send-URI	status-message	job-status attributes	status-code
Validate-Job	status-message		status-code
Cancel-Job	status-message		status-code
Get-Attributes	status-message	requested attributes	status-code
Get-Jobs	status-message	requested attributes (see the Note below)	status-code

588 Note for Get-Jobs: there is a separate attribute-sequence containing requested-attributes for each job object in the response

589 The following table shows the mapping of all IPP model document error response parameters to a parameter-sequence,
590 attribute-sequence or special position in the protocol. Those operations omitted don't have special parameters for an error
591 return.

Operation	parameter-sequence	attribute-sequence	special position
Print-Job, Print-URI, Validate-Job, Create-Job, Send-Document or Send-URI	status-message	unsupported attributes	status-code

592