Internet Printing Protocol/1.1: Encoding and Transport

Abstract

The Internet Printing Protocol (IPP) is an application-level protocol for distributed printing using Internet tools and technologies. This document defines the rules for encoding IPP operations, attributes, and values into the Internet MIME media type called "application/ipp". It also defines the rules for transporting a message body whose Content-Type is "application/ipp" over HTTP and/or HTTPS. The IPP data model and operation semantics are described in "Internet Printing Protocol/1.1: Model and Semantics" (RFC 8011).

This document obsoletes RFCs 2910 and 3382.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8010.
Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.  Introduction ......................................................... 4
2.  Conventions Used in This Document ............................... 5
  2.1.  Requirements Language .................................. 5
  2.2.  Printing Terminology .................................. 5
  2.3.  Abbreviations ........................................... 6
3.  Encoding of the Operation Layer ................................. 6
  3.1.  Picture of the Encoding .................................. 8
  3.1.1.  Request and Response .................................. 8
  3.1.2.  Attribute Group ....................................... 9
  3.1.3.  Attribute ............................................ 9
  3.1.4.  Attribute-with-one-value ............................... 10
  3.1.5.  Additional-value ....................................... 11
  3.1.6.  Collection Attribute .................................. 12
  3.1.7.  Member Attributes .................................... 13
  3.1.8.  Alternative Picture of the Encoding of a Request or a Response ........................................ 14
  3.2.  Syntax of Encoding ........................................... 15
  3.3.  Attribute-group ........................................... 16
  3.4.  Required Parameters ....................................... 18
  3.4.1.  "version-number" ..................................... 18
  3.4.2.  "operation-id" ......................................... 18
  3.4.3.  "status-code" ......................................... 19
  3.4.4.  "request-id" ......................................... 19
  3.5.  Tags ......................................................... 19
  3.5.1.  "delimiter-tag" Values ................................ 19
  3.5.2.  "value-tag" Values .................................... 20
  3.6.  "name-length" ............................................ 23
  3.7.  (Attribute) "name" ....................................... 23
  3.8.  "value-length" ............................................ 23
  3.9.  (Attribute) "value" ....................................... 24
  3.10. Data .......................................................... 25
4. Encoding of Transport Layer .................. 26
   4.1. Printer URI, Job URI, and Job ID ........... 26
5. IPP URI Schemes ............................... 28
6. IANA Considerations ............................ 29
7. Internationalization Considerations .......... 31
8. Security Considerations ....................... 31
   8.1. Security Conformance Requirements ......... 31
      8.1.1. Digest Authentication .................. 32
      8.1.2. Transport Layer Security (TLS) ......... 32
   8.2. Using IPP with TLS .......................... 33
9. Interoperability with Other IPP Versions ...... 33
   9.1. The "version-number" Parameter .............. 34
   9.2. Security and URI Schemes .................... 34
10. Changes since RFC 2910 ........................ 35
11. References ...................................... 36
    11.1. Normative References ....................... 36
    11.2. Informative References ..................... 38
Appendix A. Protocol Examples ..................... 40
   A.1. Print-Job Request ............................ 40
   A.2. Print-Job Response (Successful) ............. 41
   A.3. Print-Job Response (Failure) ................. 42
   A.4. Print-Job Response (Success with Attributes Ignored) .. 43
   A.5. Print-URI Request ............................. 45
   A.6. Create-Job Request ........................... 46
   A.7. Create-Job Request with Collection Attributes .... 46
   A.8. Get-Jobs Request ............................. 48
   A.9. Get-Jobs Response ............................ 49
Acknowledgements .................................... 51
Authors’ Addresses ................................. 51
1. Introduction

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

The transport layer consists of an HTTP request and response. All IPP implementations support HTTP/1.1, the relevant parts of which are described in the following RFCs:

- Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing [RFC7230]
- Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content [RFC7231]
- Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests [RFC7232]
- Hypertext Transfer Protocol (HTTP/1.1): Caching [RFC7234]
- Hypertext Transfer Protocol (HTTP/1.1): Authentication [RFC7235]
- The 'Basic' HTTP Authentication Scheme [RFC7617]
- HTTP Digest Access Authentication [RFC7616]

IPP implementations can support HTTP/2, which is described in the following RFCs:

- Hypertext Transfer Protocol Version 2 (HTTP/2) [RFC7540]
- HPACK - Header Compression for HTTP/2 [RFC7541]

This document specifies the HTTP headers that an IPP implementation supports.

The operation layer consists of a message body in an HTTP request or response. The "Internet Printing Protocol/1.1: Model and Semantics" document [RFC8011] and subsequent extensions (collectively known as the IPP Model) define the semantics of such a message body and the supported values. This document specifies the encoding of an IPP request and response message.
2. Conventions Used in This Document

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.2. Printing Terminology

Client: Initiator of outgoing IPP session requests and sender of outgoing IPP operation requests (Hypertext Transfer Protocol -- HTTP/1.1 [RFC7230] User Agent).

Document: An object created and managed by a Printer that contains description, processing, and status information. A Document object may have attached data and is bound to a single Job.

‘ipp’ URI: An IPP URI as defined in [RFC3510].

‘ipps’ URI: An IPPS URI as defined in [RFC7472].

Job: An object created and managed by a Printer that contains description, processing, and status information. The Job also contains zero or more Document objects.

Logical Device: A print server, software service, or gateway that processes Jobs and either forwards or stores the processed Job or uses one or more Physical Devices to render output.

Model: The semantics of operations, attributes, values, and status-codes used in the Internet Printing Protocol as defined in the Internet Printing Protocol/1.1: Model and Semantics document [RFC8011] and subsequent extensions.

Output Device: A single Logical or Physical Device.

Physical Device: A hardware implementation of an endpoint device, e.g., a marking engine, a fax modem, etc.

Printer: Listener for incoming IPP session requests and receiver of incoming IPP operation requests (Hypertext Transfer Protocol -- HTTP/1.1 [RFC7230] Server) that represents one or more Physical Devices or a Logical Device.
2.3. Abbreviations

ABNF: Augmented Backus-Naur Form [RFC5234]
HTTP: Hypertext Transfer Protocol [RFC7230]
HTTPS: HTTP over TLS [RFC2818]
IANA: Internet Assigned Numbers Authority
IEEE: Institute of Electrical and Electronics Engineers
IESG: Internet Engineering Steering Group
IPP: Internet Printing Protocol (this document and [PWG5100.12])
ISTO: IEEE Industry Standards and Technology Organization
LPD: Line Printer Daemon Protocol [RFC1179]
PWG: IEEE-ISTO Printer Working Group
RFC: Request for Comments
TCP: Transmission Control Protocol [RFC793]
TLS: Transport Layer Security [RFC5246]
URI: Uniform Resource Identifier [RFC3986]
URL: Uniform Resource Locator [RFC3986]
UTF-8: Unicode Transformation Format - 8-bit [RFC3629]

3. Encoding of the Operation Layer

The operation layer is the message body part of the HTTP request or response and it MUST contain a single IPP operation request or IPP operation response. Each request or response consists of a sequence of values and attribute groups. Attribute groups consist of a sequence of attributes each of which is a name and value. Names and values are ultimately sequences of octets.

The encoding consists of octets as the most primitive type. There are several types built from octets, but three important types are integers, character strings, and octet strings, on which most other
data types are built. Every character string in this encoding MUST be a sequence of characters where the characters are associated with some charset \[RFC2978\] and some natural language. A character string MUST be in "reading order" with the first character in the value (according to reading order) being the first character in the encoding. A character string whose associated charset is US-ASCII and whose associated natural language is US English is henceforth called a US-ASCII-STRING. A character string whose associated charset and natural language are specified in a request or response as described in the Model is henceforth called a LOCALIZED-STRING. An octet string MUST be in "Model order" with the first octet in the value (according to the Model order) being the first octet in the encoding. Every integer in this encoding MUST be encoded as a signed integer using two’s-complement binary encoding with big-endian format (also known as "network order" and "most significant byte first"). The number of octets for an integer MUST be 1, 2, or 4, depending on usage in the protocol. A one-octet integer, henceforth called a SIGNED-BYTE, is used for the version-number and tag fields. A two-byte integer, henceforth called a SIGNED-SHORT, is used for the operation-id, status-code, and length fields. A four-byte integer, henceforth called a SIGNED-INTEGER, is used for value fields and the request-id.

The following two sections present the encoding of the operation layer in two ways:

- informally through pictures and description
- formally through Augmented Backus-Naur Form (ABNF), as specified by \[RFC 5234\] [RFC5234]

An operation request or response MUST use the encoding described in these two sections.
3.1. Picture of the Encoding

3.1.1. Request and Response

An operation request or response is encoded as follows:

| ----------------------------------------------- |       |
| version-number | 2 bytes - required |
| operation-id (request) | 2 bytes - required |
| or | |
| status-code (response) | 2 bytes - required |
| request-id | 4 bytes - required |
| attribute-group | n bytes - 0 or more |
| end-of-attributes-tag | 1 byte - required |
| data | q bytes - optional |

Figure 1: IPP Message Format

The first three fields in the above diagram contain the value of attributes described in Section 4.1.1 of the Model and Semantics document [RFC8011].

The fourth field is the "attribute-group" field, and it occurs 0 or more times. Each "attribute-group" field represents a single group of attributes, such as an Operation Attributes group or a Job Attributes group (see the Model). The Model specifies the required attribute groups and their order for each operation request and response.

The "end-of-attributes-tag" field is always present, even when the "data" is not present. The Model specifies whether the "data" field is present for each operation request and response.
3.1.2. Attribute Group

Each "attribute-group" field is encoded as follows:

```
|          begin-attribute-group-tag          |  1 byte
|--------------------------------------------|
|             attribute                      |  p bytes |- 0 or more
```

Figure 2: Attribute Group Encoding

An "attribute-group" field contains zero or more "attribute" fields.

Note that the values of the "begin-attribute-group-tag" field and the "end-of-attributes-tag" field are called "delimiter-tags".

3.1.3. Attribute

An "attribute" field is encoded as follows:

```
|          attribute-with-one-value          |  q bytes
|-------------------------------------------|
|             additional-value              |  r bytes |- 0 or more
```

Figure 3: Attribute Encoding

When an attribute is single valued (e.g., "copies" with a value of 10) or multi-valued with one value (e.g., "sides-supported" with just the value 'one-sided'), it is encoded with just an "attribute-with-one-value" field. When an attribute is multi-valued with n values (e.g., "sides-supported" with the values 'one-sided' and 'two-sided-long-edge'), it is encoded with an "attribute-with-one-value" field followed by n-1 "additional-value" fields.
3.1.4. Attribute-with-one-value

Each "attribute-with-one-value" field is encoded as follows:

```
|                    value-tag                  |  1 byte
|---------------------------------------------|
|                name-length (value is u)     |  2 bytes
|---------------------------------------------|
|                         name                |  u bytes
|---------------------------------------------|
|                value-length (value is v)     |  2 bytes
|---------------------------------------------|
|                         value              |  v bytes
```

Figure 4: Single Value Attribute Encoding

An "attribute-with-one-value" field is encoded with five subfields:

- The "value-tag" field specifies the attribute syntax, e.g., 0x44 for the attribute syntax ‘keyword’.
- The "name-length" field specifies the length of the "name" field in bytes, e.g., u in the above diagram or 15 for the name "sides-supported".
- The "name" field contains the textual name of the attribute, e.g., "sides-supported".
- The "value-length" field specifies the length of the "value" field in bytes, e.g., v in the above diagram or 9 for the (keyword) value ‘one-sided’.
- The "value" field contains the value of the attribute, e.g., the textual value ‘one-sided’.
3.1.5. Additional-value

Each "additional-value" field is encoded as follows:

-----------------------------------------------
|                   value-tag                 |   1 byte
-----------------------------------------------
|            name-length  (value is 0x0000)   |   2 bytes
-----------------------------------------------
|              value-length (value is w)      |   2 bytes
-----------------------------------------------
|                     value                   |   w bytes
-----------------------------------------------

Figure 5: Additional Attribute Value Encoding

An "additional-value" is encoded with four subfields:

- The "value-tag" field specifies the attribute syntax, e.g., 0x44 for the attribute syntax ‘keyword’.

- The "name-length" field has the value of 0 in order to signify that it is an "additional-value". The value of the "name-length" field distinguishes an "additional-value" field ("name-length" is 0) from an "attribute-with-one-value" field ("name-length" is not 0).

- The "value-length" field specifies the length of the "value" field in bytes, e.g., w in the above diagram or 19 for the (keyword) value ‘two-sided-long-edge’.

- The "value" field contains the value of the attribute, e.g., the textual value ‘two-sided-long-edge’.
3.1.6. Collection Attribute

Collection attributes create a named group containing related "member" attributes. The "attribute-with-one-value" field for a collection attribute is encoded as follows:

```
<table>
<thead>
<tr>
<th>value-tag (value is 0x34)</th>
<th>1 byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-length (value is u)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>name</td>
<td>u bytes</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>value-length (value is 0x0000)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>member-attribute</td>
<td>q bytes</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>end-value-tag (value is 0x37)</td>
<td>1 byte</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>end-name-length (value is 0x0000)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>end-value-length (value is 0x0000)</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>
```

Figure 6: Collection Attribute Encoding

Collection attribute is encoded with eight subfields:

- The "value-tag" field specifies the start attribute syntax: 0x34 for the attribute syntax 'begCollection'.
- The "name-length" field specifies the length of the "name" field in bytes, e.g., u in the above diagram or 9 for the name "media-col". Additional collection attribute values use a name length of 0x0000.
- The "name" field contains the textual name of the attribute, e.g., "media-col".
- The "value-length" field specifies a length of 0x0000.
- The "member-attribute" field contains member attributes encoded as defined in Section 3.1.7.
- The "end-value-tag" field specifies the end attribute syntax: 0x37 for the attribute syntax 'endCollection'.
- The "end-name-length" field specifies a length of 0x0000.
3.1.7. Member Attributes

Each "member-attribute" field is encoded as follows:

```
|          value-tag (value is 0x4a)          |   1 byte
|        name-length (value is 0x0000)        |   2 bytes
|          value-length (value is w)          |   2 bytes
|          value (member-name)                |   w bytes
|        member-value-tag                    |   1 byte
|        name-length (value is 0x0000)        |   2 bytes
|    member-value-length (value is x)         |   2 bytes
|        member-value                         |   x bytes
```

Figure 7: Member Attribute Encoding

A "member-attribute" is encoded with eight subfields:

- The "value-tag" field specifies 0x4a for the attribute syntax 'memberAttrName'.

- The "name-length" field has the value of 0 in order to signify that it is a "member-attribute" contained in the collection.

- The "value-length" field specifies the length of the "value" field in bytes, e.g., w in the above diagram or 10 for the member attribute name 'media-type'. Additional member attribute values are specified using a value length of 0.

- The "value" field contains the name of the member attribute, e.g., the textual value 'media-type'.

- The "member-value-tag" field specifies the attribute syntax for the member attribute, e.g., 0x44 for the attribute syntax 'keyword'.
The second "name-length" field has the value of 0 in order to signify that it is a "member-attribute" contained in the collection.

The "member-value-length" field specifies the length of the member attribute value, e.g., x in the above diagram or 10 for the value 'stationery'.

The "member-value" field contains the value of the attribute, e.g., the textual value 'stationery'.

3.1.8. Alternative Picture of the Encoding of a Request or a Response

From the standpoint of a parser that performs an action based on a "tag" value, the encoding consists of:

```
<table>
<thead>
<tr>
<th>version-number</th>
<th>2 bytes - required</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation-id (request)</td>
<td>2 bytes - required</td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>status-code (response)</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>request-id</td>
<td>4 bytes - required</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>tag (delimiter-tag or value-tag)</td>
<td>1 byte</td>
</tr>
<tr>
<td>empty or rest of attribute</td>
<td>x bytes</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>end-of-attributes-tag</td>
<td>1 byte - required</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>data</td>
<td>y bytes - optional</td>
</tr>
</tbody>
</table>
```

Figure 8: Encoding Based on Value Tags

The following shows what fields the parser would expect after each type of "tag":

- "begin-attribute-group-tag": expect zero or more "attribute" fields
- "value-tag": expect the remainder of an "attribute-with-one-value" or an "additional-value"
- "end-of-attributes-tag": expect that "attribute" fields are complete and there is optional "data"
3.2. Syntax of Encoding

The ABNF [RFC5234] syntax for an IPP message is shown in Figure 9.

ipp-message = ipp-request / ipp-response
ipp-request = version-number operation-id request-id
              *attribute-group end-of-attributes-tag data
ipp-response = version-number status-code request-id
               *attribute-group end-of-attributes-tag data

version-number = major-version-number minor-version-number
major-version-number = SIGNED-BYTE
minor-version-number = SIGNED-BYTE

operation-id = SIGNED-SHORT ; mapping from model
status-code = SIGNED-SHORT ; mapping from model
request-id = SIGNED-INTEGER ; whose value is > 0

attribute-group = begin-attribute-group-tag *attribute
attribute = attribute-with-one-value *additional-value
attribute-with-one-value = value-tag name-length name
                          value-length value
additional-value = value-tag zero-name-length
                   value-length value

name-length = SIGNED-SHORT ; number of octets of 'name'
name = LALPHA *(( LALPHA / DIGIT / "-" / "_" / "." )
value-length = SIGNED-SHORT ; number of octets of 'value'
value = OCTET-STRING
data = OCTET-STRING

zero-name-length = %x00.00 ; name-length of 0
value-tag = %x10-ff ; see Section 3.5.2
begin-attribute-group-tag = %x00-02 / %x04-0f ; see Section 3.5.1
end-of-attributes-tag = %x03 ; tag of 3
                        ; see Section 3.5.1

SIGNED-BYTE = BYTE
SIGNED-SHORT = 2BYTE
SIGNED-INTEGER = 4BYTE
DIGIT = %x30-39 ; "0" to "9"
LALPHA = %x61-7a ; "a" to "z"
BYTE = %x00-ff
OCTET-STRING = *BYTE

Figure 9: ABNF of IPP Message Format
Figure 10 defines additional terms that are referenced in this document and provides an alternate grouping of the delimiter tags.

delimiter-tag = begin-attribute-group-tag / ; see Section 3.5.1
end-of-attributes-tag
begin-attribute-group-tag = %x00 / operation-attributes-tag /
job-attributes-tag / printer-attributes-tag /
unsupported-attributes-tag / future-group-tags
operation-attributes-tag   = %x01             ; tag of 1
job-attributes-tag         = %x02             ; tag of 2
end-of-attributes-tag      = %x03             ; tag of 3
printer-attributes-tag     = %x04             ; tag of 4
unsupported-attributes-tag = %x05             ; tag of 5
future-group-tags          = %x06-0f          ; future extensions

Figure 10: ABNF for Attribute Group Tags

3.3. Attribute-group

Each "attribute-group" field MUST be encoded with the "begin-
attribute-group-tag" field followed by zero or more "attribute" sub-
fields.
Table 1 maps the Model group name to value of the "begin-attribute-group-tag" field:

<table>
<thead>
<tr>
<th>Model Document Group</th>
<th>&quot;begin-attribute-group-tag&quot; field values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Attributes</td>
<td>&quot;operations-attributes-tag&quot;</td>
</tr>
<tr>
<td>Job Template Attributes</td>
<td>&quot;job-attributes-tag&quot;</td>
</tr>
<tr>
<td>Job Object Attributes</td>
<td>&quot;job-attributes-tag&quot;</td>
</tr>
<tr>
<td>Unsupported Attributes</td>
<td>&quot;unsupported-attributes-tag&quot;</td>
</tr>
<tr>
<td>Requested Attributes</td>
<td>(Get-Job-Attributes) &quot;job-attributes-tag&quot;</td>
</tr>
<tr>
<td>Requested Attributes</td>
<td>(Get-Printer-Attributes)&quot;printer-attributes-tag&quot;</td>
</tr>
<tr>
<td>Document Content</td>
<td>in a special position at the end of the message as described in Section 3.1.1.</td>
</tr>
</tbody>
</table>

Table 1: Group Values

For each operation request and response, the Model prescribes the required and optional attribute groups, along with their order. Within each attribute group, the Model prescribes the required and optional attributes, along with their order.

When the Model requires an attribute group in a request or response and the attribute group contains zero attributes, a request or response SHOULD encode the attribute group with the "begin-attribute-group-tag" field followed by zero "attribute" fields. For example, if the Client requests a single unsupported attribute with the Get-Printer-Attributes operation, the Printer MUST return no "attribute" fields, and it SHOULD return a "begin-attribute-group-tag" field for the Printer Attributes group. The Unsupported Attributes group is not such an example. According to the Model, the Unsupported Attributes group SHOULD be present only if the Unsupported Attributes group contains at least one attribute.
A receiver of a request MUST be able to process the following as equivalent empty attribute groups:

a. A "begin-attribute-group-tag" field with zero following "attribute" fields.

b. A missing, but expected, "begin-attribute-group-tag" field.

When the Model requires a sequence of an unknown number of attribute groups, each of the same type, the encoding MUST contain one "begin-attribute-group-tag" field for each attribute group, even when an "attribute-group" field contains zero "attribute" sub-fields. For example, the Get-Jobs operation may return zero attributes for some Jobs and not others. The "begin-attribute-group-tag" field followed by zero "attribute" fields tells the recipient that there is a Job in queue for which no information is available except that it is in the queue.

3.4. Required Parameters

Some operation elements are called parameters in the Model. They MUST be encoded in a special position and they MUST NOT appear as operation attributes. These parameters are described in the subsections below.

3.4.1. "version-number"

The "version-number" field consists of a major and minor version-number, each of which is represented by a SIGNED-BYTE. The major version-number is the first byte of the encoding and the minor version-number is the second byte of the encoding. The protocol described in [RFC8011] has a major version-number of 1 (0x01) and a minor version-number of 1 (0x01). The ABNF for these two bytes is %x01.01.

Note: See Section 9 for more information on the "version-number" field and IPP version numbers.

3.4.2. "operation-id"

The "operation-id" field contains an operation-id value as defined in the Model. The value is encoded as a SIGNED-SHORT and is located in the third and fourth bytes of the encoding of an operation request.
3.4.3.  "status-code"

The "status-code" field contains a status-code value as defined in the Model. The value is encoded as a SIGNED-SHORT and is located in the third and fourth bytes of the encoding of an operation response.

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

3.4.4.  "request-id"

The "request-id" field contains the request-id value as defined in the Model. The value is encoded as a SIGNED-INTEGER and is located in the fifth through eighth bytes of the encoding.

3.5.  Tags

There are two kinds of tags:

- delimiter tags: delimit major sections of the protocol, namely attribute groups and data
- value tags: specify the type of each attribute value

Tags are part of the IANA IPP registry [IANA-IPP]

3.5.1.  "delimiter-tag" Values

Table 2 specifies the values for the delimiter tags defined in this document. These tags are registered, along with tags defined in other documents, in the "Attribute Group Tags" registry.

<table>
<thead>
<tr>
<th>Tag Value (Hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x01</td>
<td>&quot;operation-attributes-tag&quot;</td>
</tr>
<tr>
<td>0x02</td>
<td>&quot;job-attributes-tag&quot;</td>
</tr>
<tr>
<td>0x03</td>
<td>&quot;end-of-attributes-tag&quot;</td>
</tr>
<tr>
<td>0x04</td>
<td>&quot;printer-attributes-tag&quot;</td>
</tr>
<tr>
<td>0x05</td>
<td>&quot;unsupported-attributes-tag&quot;</td>
</tr>
</tbody>
</table>

Table 2: "delimiter-tag" Values
When a "begin-attribute-group-tag" field occurs in the protocol, it means that zero or more following attributes up to the next group tag are attributes belonging to the attribute group specified by the value of the "begin-attribute-group-tag". For example, if the value of "begin-attribute-group-tag" is 0x01, the following attributes are members of the Operations Attributes group.

The "end-of-attributes-tag" (value 0x03) MUST occur exactly once in an operation and MUST be the last "delimiter-tag". If the operation has a document-data group, the Document data in that group follows the "end-of-attributes-tag".

The order and presence of "attribute-group" fields (whose beginning is marked by the "begin-attribute-group-tag" subfield) for each operation request and each operation response MUST be that defined in the Model.

A Printer MUST treat a "delimiter-tag" (values from 0x00 through 0x0f) differently from a "value-tag" (values from 0x10 through 0xff) so that the Printer knows there is an entire attribute group as opposed to a single value.

3.5.2. "value-tag" Values

The remaining tables show values for the "value-tag" field, which is the first octet of an attribute. The "value-tag" field specifies the type of the value of the attribute.

Table 3 specifies the "out-of-band" values for the "value-tag" field defined in this document. These tags are registered, along with tags defined in other documents, in the "Out-of-Band Attribute Value Tags" registry.

<table>
<thead>
<tr>
<th>Tag Value (Hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10</td>
<td>unsupported</td>
</tr>
<tr>
<td>0x12</td>
<td>unknown</td>
</tr>
<tr>
<td>0x13</td>
<td>no-value</td>
</tr>
</tbody>
</table>

Table 3: Out-of-Band Values
Table 4 specifies the integer values defined in this document for the "value-tag" field; they are registered in the "Attribute Syntaxes" registry.

<table>
<thead>
<tr>
<th>Tag Value (Hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20</td>
<td>Unassigned integer data type (see IANA IPP registry)</td>
</tr>
<tr>
<td>0x21</td>
<td>integer</td>
</tr>
<tr>
<td>0x22</td>
<td>boolean</td>
</tr>
<tr>
<td>0x23</td>
<td>enum</td>
</tr>
<tr>
<td>0x24-0x2f</td>
<td>Unassigned integer data types (see IANA IPP registry)</td>
</tr>
</tbody>
</table>

Table 4: Integer Tags

Table 5 specifies the octetString values defined in this document for the "value-tag" field; they are registered in the "Attribute Syntaxes" registry.

<table>
<thead>
<tr>
<th>Tag Value (Hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30</td>
<td>octetString with an unspecified format</td>
</tr>
<tr>
<td>0x31</td>
<td>dateTime</td>
</tr>
<tr>
<td>0x32</td>
<td>resolution</td>
</tr>
<tr>
<td>0x33</td>
<td>rangeOfInteger</td>
</tr>
<tr>
<td>0x34</td>
<td>begCollection</td>
</tr>
<tr>
<td>0x35</td>
<td>textWithLanguage</td>
</tr>
<tr>
<td>0x36</td>
<td>nameWithLanguage</td>
</tr>
<tr>
<td>0x37</td>
<td>endCollection</td>
</tr>
<tr>
<td>0x38-0x3f</td>
<td>Unassigned octetString data types (see IANA IPP registry)</td>
</tr>
</tbody>
</table>

Table 5: octetString Tags
Table 6 specifies the character-string values defined in this document for the "value-tag" field; they are registered in the "Attribute Syntaxes" registry.

<table>
<thead>
<tr>
<th>Tag Value (Hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>Unassigned character-string data type (see IANA IPP registry)</td>
</tr>
<tr>
<td>0x41</td>
<td>textWithoutLanguage</td>
</tr>
<tr>
<td>0x42</td>
<td>nameWithoutLanguage</td>
</tr>
<tr>
<td>0x43</td>
<td>Unassigned character-string data type (see IANA IPP registry)</td>
</tr>
<tr>
<td>0x44</td>
<td>keyword</td>
</tr>
<tr>
<td>0x45</td>
<td>uri</td>
</tr>
<tr>
<td>0x46</td>
<td>uriScheme</td>
</tr>
<tr>
<td>0x47</td>
<td>charset</td>
</tr>
<tr>
<td>0x48</td>
<td>naturallanguage</td>
</tr>
<tr>
<td>0x49</td>
<td>mimeMediaType</td>
</tr>
<tr>
<td>0x4a</td>
<td>memberAttrName</td>
</tr>
<tr>
<td>0x4b-0x5f</td>
<td>Unassigned character-string data types (see IANA IPP registry)</td>
</tr>
</tbody>
</table>

Table 6: String Tags

Note: An attribute value always has a type, which is explicitly specified by its tag; one such tag value is "nameWithoutLanguage". An attribute’s name has an implicit type, which is keyword.

The values 0x60-0xff are reserved for future type definitions in Standards Track documents.

The tag 0x7f is reserved for extending types beyond the 255 values available with a single byte. A tag value of 0x7f MUST signify that the first four bytes of the value field are interpreted as the tag value. Note this future extension doesn’t affect parsers that are unaware of this special tag. The tag is like any other unknown tag, and the value length specifies the length of a value, which contains a value that the parser treats atomically. Values from 0x00000000 to 0xFFFFFFFF are reserved for definition in future Standards Track documents. The values 0x40000000 to 0x7fffffff are reserved for vendor extensions.
3.6. "name-length"

The "name-length" field consists of a SIGNED-SHORT and specifies the number of octets in the immediately following "name" field. The value of this field excludes the two bytes of the "name-length" field. For example, if the "name" field contains 'sides', the value of this field is 5.

If a "name-length" field has a value of zero, the following "name" field is empty and the following value is treated as an additional value for the attribute encoded in the nearest preceding "attribute-with-one-value" field. Within an attribute group, if two or more attributes have the same name, the attribute group is malformed (see [RFC8011]). The zero-length name is the only mechanism for multi-valued attributes.

3.7. (Attribute) "name"

The "name" field contains the name of an attribute. The Model specifies such names.

3.8. "value-length"

The "value-length" field consists of a SIGNED-SHORT, which specifies the number of octets in the immediately following "value" field. The value of this field excludes the two bytes of the "value-length" field. For example, if the "value" field contains the keyword (string) value 'one-sided', the value of this field is 9.

For any of the types represented by binary signed integers, the sender MUST encode the value in exactly four octets.

For any of the types represented by binary signed bytes, e.g., the boolean type, the sender MUST encode the value in exactly one octet.

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

For "out-of-band" values for the "value-tag" field defined in this document, such as 'unsupported', the "value-length" MUST be 0 and the "value" empty; the "value" has no meaning when the "value-tag" has one of these "out-of-band" values. For future "out-of-band" "value-tag" fields, the same rule holds unless the definition explicitly states that the "value-length" MAY be non-zero and the "value" non-empty.
### 3.9. (Attribute) "value"

The syntax types (specified by the "value-tag" field) and most of the details of the representation of attribute values are defined in the Model. Table 7 augments the information in the Model and defines the syntax types from the Model in terms of the five basic types defined in Section 3. The five types are US-ASCII-STRING, LOCALIZED-STRING, SIGNED-INTEGER, SIGNED-SHORT, SIGNED-BYTE, and OCTET-STRING.

<table>
<thead>
<tr>
<th>Syntax of Attribute Value</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>textWithoutLanguage, nameWithoutLanguage</td>
<td>LOCALIZED-STRING</td>
</tr>
<tr>
<td>textWithLanguage</td>
<td>OCTET-STRING consisting of four fields: a SIGNED-SHORT, which is the number of octets in the following field; a value of type natural-language; a SIGNED-SHORT, which is the number of octets in the following field; and a value of type textWithoutLanguage. The length of a textWithLanguage value MUST be 4 + the value of field a + the value of field c.</td>
</tr>
<tr>
<td>nameWithLanguage</td>
<td>OCTET-STRING consisting of four fields: a SIGNED-SHORT, which is the number of octets in the following field; a value of type natural-language; a SIGNED-SHORT, which is the number of octets in the following field; and a value of type nameWithoutLanguage. The length of a nameWithLanguage value MUST be 4 + the value of field a + the value of field c.</td>
</tr>
<tr>
<td>charset, naturalLanguage, mimeMediaType, keyword, uri, and uriScheme</td>
<td>US-ASCII-STRING</td>
</tr>
<tr>
<td>boolean</td>
<td>SIGNED-BYTE where 0x00 is 'false' and 0x01 is 'true'</td>
</tr>
<tr>
<td>integer and enum</td>
<td>a SIGNED-INTEGER</td>
</tr>
</tbody>
</table>
### Table 7: Attribute Value Encoding

The attribute syntax type of the value determines its encoding and the value of its "value-tag".

#### 3.10. Data

The "data" field MUST include any data required by the operation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateTime</td>
<td>OCTET-STRING consisting of eleven octets whose contents are defined by</td>
</tr>
<tr>
<td></td>
<td>&quot;DateAndTime&quot; in RFC 2579 [RFC2579]</td>
</tr>
<tr>
<td>resolution</td>
<td>OCTET-STRING consisting of nine octets of two SIGNED-INTEGERs followed</td>
</tr>
<tr>
<td></td>
<td>by a SIGNED-BYTE. The first SIGNED-INTEGER contains the value of cross-</td>
</tr>
<tr>
<td></td>
<td>feed direction resolution. The second SIGNED-INTEGER contains the value</td>
</tr>
<tr>
<td></td>
<td>of feed direction resolution. The SIGNED-BYTE contains the units value.</td>
</tr>
<tr>
<td>rangeOfInteger</td>
<td>Eight octets consisting of two SIGNED-INTEGERs. The first SIGNED-</td>
</tr>
<tr>
<td></td>
<td>INTEGER contains the lower bound and the second SIGNED-INTEGER contains</td>
</tr>
<tr>
<td></td>
<td>the upper bound.</td>
</tr>
<tr>
<td>1setOf X</td>
<td>Encoding according to the rules for an attribute with more than one</td>
</tr>
<tr>
<td></td>
<td>value. Each value X is encoded according to the rules for encoding its</td>
</tr>
<tr>
<td></td>
<td>type.</td>
</tr>
<tr>
<td>octetString</td>
<td>OCTET-STRING</td>
</tr>
<tr>
<td>collection</td>
<td>Encoding as defined in Section 3.1.6.</td>
</tr>
</tbody>
</table>
4. Encoding of Transport Layer

HTTP/1.1 [RFC7230] is the REQUIRED transport layer for this protocol. HTTP/2 [RFC7540] is an OPTIONAL transport layer for this protocol.

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

- the target URI for the operation; and
- the total length of the data in the operation layer, either as a single length or as a sequence of chunks each with a length.

Printer implementations MUST support HTTP over the IANA-assigned well-known port 631 (the IPP default port), although a Printer implementation can support HTTP over some other port as well.

Each HTTP operation MUST use the POST method where the request-target is the object target of the operation and where the "Content-Type" of the message body in each request and response MUST be "application/ipp". The message body MUST contain the operation layer and MUST have the syntax described in Section 3.2, "Syntax of Encoding". A Client implementation MUST adhere to the rules for a Client described for HTTP [RFC7230]. A Printer (server) implementation MUST adhere to the rules for an origin server described for HTTP [RFC7230].

An IPP server sends a response for each request that it receives. If an IPP server detects an error, it MAY send a response before it has read the entire request. If the HTTP layer of the IPP server completes processing the HTTP headers successfully, it MAY send an intermediate response, such as "100 Continue", with no IPP data before sending the IPP response. A Client MUST expect such a variety of responses from an IPP server. For further information on HTTP, consult the HTTP documents [RFC7230].

An HTTP/1.1 server MUST support chunking for IPP requests, and an IPP Client MUST support chunking for IPP responses according to HTTP/1.1 [RFC7230].

4.1. Printer URI, Job URI, and Job ID

All Printer and Job objects are identified by a Uniform Resource Identifier (URI) [RFC3986] so that they can be persistently and unambiguously referenced. Jobs can also be identified by a combination of Printer URI and Job ID.
Some operation elements are encoded twice, once as the request-target on the HTTP request-line and a second time as a REQUIRED operation attribute in the application/ipp entity. These attributes are the target for the operation and are called "printer-uri" and "job-uri".

Note: The target URI is included twice in an operation referencing the same IPP object, but the two URIs can be different. For example, the HTTP request-target can be relative while the IPP request URI is absolute.

HTTP allows Clients to generate and send a relative URI rather than an absolute URI. A relative URI identifies a resource with the scope of the HTTP server but does not include scheme, host, or port. The following statements characterize how URIs are used in the mapping of IPP onto HTTP:

1. Although potentially redundant, a Client MUST supply the target of the operation both as an operation attribute and as a URI at the HTTP layer. The rationale for this decision is to maintain a consistent set of rules for mapping "application/ipp" to possibly many communication layers, even where URIs are not used as the addressing mechanism in the transport layer.

2. Even though these two URIs might not be literally identical (one being relative and the other being absolute), they MUST both reference the same IPP object.

3. The URI in the HTTP layer is either relative or absolute and is used by the HTTP server to route the HTTP request to the correct resource relative to that HTTP server.

4. Once the HTTP server resource begins to process the HTTP request, it can get the reference to the appropriate IPP Printer object from either the HTTP URI (using to the context of the HTTP server for relative URIs) or from the URI within the operation request; the choice is up to the implementation.

5. HTTP URIs can be relative or absolute, but the target URI in the IPP operation attribute MUST be an absolute URI.
5. IPP URI Schemes

The IPP URI schemes are ’ipp’ [RFC3510] and ’ipps’ [RFC7472]. Clients and Printers MUST support the ipp-URI value in the following IPP attributes:

- Job attributes:
  - * job-uri
  - * job-printer-uri
- Printer attributes:
  - * printer-uri-supported
- Operation attributes:
  - * job-uri
  - * printer-uri

Each of the above attributes identifies a Printer or Job. The ipp-URI and ipps-URI are intended as the value of the attributes in this list. All of these attributes have a syntax type of ‘uri’, but there are attributes with a syntax type of ‘uri’ that do not use the ‘ipp’ scheme, e.g., "job-more-info".

If a Printer registers its URI with a directory service, the Printer MUST register an ipp-URI or ipps-URI.

When a Client sends a request, it MUST convert a target ipp-URI to a target http-URL (or ipps-URI to a target https-URL) for the HTTP layer according to the following steps:

1. change the ‘ipp’ scheme to ‘http’ or ‘ipps’ scheme to ‘https’; and

2. add an explicit port 631 if the ipp-URL or ipps-URL does not contain an explicit port. Note that port 631 is the IANA-assigned well-known port for the ‘ipp’ and ‘ipps’ schemes.

The Client MUST use the target http-URL or https-URL in both the HTTP request-line and HTTP headers, as specified by HTTP [RFC7230]. However, the Client MUST use the target ipp-URI or ipps-URI for the value of the "printer-uri" or "job-uri" operation attribute within the application/ipp body of the request. The server MUST use the
ipp-URI or ipps-URI for the value of the "printer-uri", "job-uri", or "printer-uri-supported" attributes within the application/ipp body of the response.

For example, when an IPP Client sends a request directly, i.e., no proxy, to an ipp-URI "ipp://printer.example.com/ipp/print/myqueue", it opens a TCP connection to port 631 (the IPP implicit port) on the host "printer.example.com" and sends the following data:

```
POST /ipp/print/myqueue HTTP/1.1
Host: printer.example.com:631
Content-type: application/ipp
Transfer-Encoding: chunked

"printer-uri" 'ipp://printer.example.com/ipp/print/myqueue'
(encoded in application/ipp message body)
```

Figure 11: Direct IPP Request

As another example, when an IPP Client sends the same request as above via a proxy "myproxy.example.com", it opens a TCP connection to the proxy port 8080 on the proxy host "myproxy.example.com" and sends the following data:

```
POST http://printer.example.com:631/ipp/print/myqueue HTTP/1.1
Host: printer.example.com:631
Content-type: application/ipp
Transfer-Encoding: chunked

"printer-uri" 'ipp://printer.example.com/ipp/print/myqueue'
(encoded in application/ipp message body)
```

Figure 12: Proxied IPP Request

The proxy then connects to the IPP origin server with headers that are the same as the "no-proxy" example above.

6. IANA Considerations

The IANA-PRINTER-MIB [RFC3805] has been updated to reference this document; the current version is available from <http://www.iana.org>.

See the IANA Considerations in the document "Internet Printing Protocol/1.1: Model and Semantics" [RFC8011] for information on IANA considerations for IPP extensions. IANA has updated the existing
'application/ipp' media type registration (whose contents are defined in Section 3 "Encoding of the Operation Layer") with the following information.

Type name: application

Subtype name: ipp

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: IPP requests/responses MAY contain long lines and ALWAYS contain binary data (for example, attribute value lengths).

Security considerations: IPP requests/responses do not introduce any security risks not already inherent in the underlying transport protocols. Protocol mixed-version interworking rules in [RFC8011] as well as protocol-encoding rules in this document are complete and unambiguous. See also the security considerations in this document and [RFC8011].

Interoperability considerations: IPP requests (generated by Clients) and responses (generated by servers) MUST comply with all conformance requirements imposed by the normative specifications [RFC8011] and this document. Protocol-encoding rules specified in RFC 8010 are comprehensive so that interoperability between conforming implementations is guaranteed (although support for specific optional features is not ensured). Both the "charset" and "natural-language" of all IPP attribute values that are a LOCALIZED-STRING are explicit within IPP requests/responses (without recourse to any external information in HTTP, SMTP, or other message transport headers).

Published specifications: RFCs 8010 and 8011

Applications that use this media type: Internet Printing Protocol (IPP) print clients and print servers that communicate using HTTP/HTTPS or other transport protocols. Messages of type "application/ipp" are self-contained and transport independent, including "charset" and "natural-language" context for any LOCALIZED-STRING value.

Fragment identifier considerations: N/A
7. Internationalization Considerations

See the section on "Internationalization Considerations" in the document "Internet Printing Protocol/1.1: Model and Semantics" [RFC8011] for information on internationalization. This document adds no additional issues.

8. Security Considerations

The IPP Model and Semantics document [RFC8011] discusses high-level security requirements (Client Authentication, Server Authentication, and Operation Privacy). Client Authentication is the mechanism by which the Client proves its identity to the server in a secure manner. Server Authentication is the mechanism by which the server proves its identity to the Client in a secure manner. Operation Privacy is defined as a mechanism for protecting operations from eavesdropping.

Message Integrity is addressed in the document "Internet Printing Protocol (IPP) over HTTPS Transport Binding and the ‘ipps’ URI Scheme" [RFC7472].

8.1. Security Conformance Requirements

This section defines the security requirements for IPP Clients and IPP objects.
8.1.1. Digest Authentication

IPP Clients and Printers SHOULD support Digest Authentication [RFC7616]. Use of the Message Integrity feature (qop="auth-int") is OPTIONAL.

Note: Previous versions of this specification required support for the MD5 algorithms; however, [RFC7616] makes SHA2-256 mandatory to implement and deprecates MD5, only allowing its use for backwards compatibility reasons. IPP implementations that support Digest Authentication MUST support SHA2-256 and SHOULD support MD5 for backwards compatibility.

Note: The reason that IPP Clients and Printers SHOULD (rather than MUST) support Digest Authentication is that there is a certain class of Output Devices where it does not make sense. Specifically, a low-end device with limited ROM space and low paper throughput may not need Client Authentication. This class of device typically requires firmware designers to make trade-offs between protocols and functionality to arrive at the lowest-cost solution possible. Factored into the designer’s decisions is not just the size of the code, but also the testing, maintenance, usefulness, and time-to-market impact for each feature delivered to the customer. Forcing such low-end devices to provide security in order to claim IPP/1.1 conformance would not make business sense. Print devices that have high-volume throughput and have available ROM space will typically provide support for Client Authentication that safeguards the device from unauthorized access because these devices are prone to a high loss of consumables and paper if unauthorized access occurs.

8.1.2. Transport Layer Security (TLS)

IPP Clients and Printers SHOULD support Transport Layer Security (TLS) [RFC5246] [RFC7525] for Server Authentication and Operation Privacy. IPP Printers MAY also support TLS for Client Authentication. IPP Clients and Printers MAY support Basic Authentication [RFC7617] for User Authentication if the channel is secure, e.g., IPP over HTTPS [RFC7472]. IPP Clients and Printers SHOULD NOT support Basic Authentication over insecure channels.

The IPP Model and Semantics document [RFC8011] defines two Printer attributes ("uri-authentication-supported" and "uri-security-supported") that the Client can use to discover the security policy of a Printer. That document also outlines IPP-specific security considerations and is the primary reference for security implications with regard to the IPP itself.
Note: Because previous versions of this specification did not require TLS support, this version cannot require it for IPP/1.1. However, since printing often involves a great deal of sensitive or private information (medical reports, performance reviews, banking information, etc.) and network monitoring is pervasive ([RFC7258]), implementors are strongly encouraged to include TLS support.

Note: Because IPP Printers typically use self-signed X.509 certificates, IPP Clients SHOULD support Trust On First Use (defined in [RFC7435]) in addition to traditional X.509 certificate validation.

8.2. Using IPP with TLS

IPP uses the "Upgrading to TLS Within HTTP/1.1" mechanism [RFC2817] for 'ipp' URIs. The Client requests a secure TLS connection by using the HTTP "Upgrade" header while the server agrees in the HTTP response. The switch to TLS occurs either because the server grants the Client’s request to upgrade to TLS or a server asks to switch to TLS in its response. Secure communication begins with a server’s response to switch to TLS.

IPP uses the "HTTPS: HTTP over TLS" mechanism [RFC2818] for 'ipps' URIs. The Client and server negotiate a secure TLS connection immediately and unconditionally.

9. Interoperability with Other IPP Versions

It is beyond the scope of this specification to mandate conformance with versions of IPP other than 1.1. IPP was deliberately designed, however, to make supporting other versions easy. IPP objects (Printers, Jobs, etc.) SHOULD:

- understand any valid request whose major "version-number" is greater than 0; and
- respond appropriately with a response containing the same "version-number" parameter value used by the Client in the request (if the Client-supplied "version-number" is supported) or the highest "version-number" supported by the Printer (if the Client-supplied "version-number" is not supported).

IPP Clients SHOULD:

- understand any valid response whose major "version-number" is greater than 0.
9.1.  The "version-number" Parameter

The following are rules regarding the "version-number" parameter (see Section 3.3):

1.  Clients MUST send requests containing a "version-number" parameter with the highest supported value, e.g., '1.1', '2.0', etc., and SHOULD try supplying alternate version numbers if they receive a 'server-error-version-not-supported' error return in a response. For example, if a Client sends an IPP/2.0 request that is rejected with the 'server-error-version-not-supported' error and an IPP/1.1 "version-number", it SHOULD retry by sending an IPP/1.1 request.

2.  IPP objects (Printers, Jobs, etc.) MUST accept requests containing a "version-number" parameter with a '1.1' value (or reject the request for reasons other than 'server-error-version-not-supported').

3.  IPP objects SHOULD either accept requests whose major version is greater than 0 or reject such requests with the 'server-error-version-not-supported' status-code. See Section 4.1.8 of [RFC8011].

4.  In any case, security MUST NOT be compromised when a Client supplies a lower "version-number" parameter in a request. For example, if an IPP/2.0 conforming Printer accepts version '1.1' requests and is configured to enforce Digest Authentication, it MUST do the same for a version '1.1' request.

9.2.  Security and URI Schemes

The following are rules regarding security, the "version-number" parameter, and the URI scheme supplied in target attributes and responses:

1.  When a Client supplies a request, the "printer-uri" or "job-uri" target operation attribute MUST have the same scheme as that indicated in one of the values of the "printer-uri-supported" Printer attribute.

2.  When the Printer returns the "job-printer-uri" or "job-uri" Job Description attributes, it SHOULD return the same scheme ('ipp', 'ipps', etc.) that the Client supplied in the "printer-uri" or "job-uri" target operation attributes in the Get-Job-Attributes or Get-Jobs request, rather than the scheme used when the Job was created. However, when a Client requests Job attributes using the Get-Job-Attributes or Get-Jobs operations, the Jobs and Job
attributes that the Printer returns depends on: (1) the security in effect when the Job was created, (2) the security in effect in the query request, and (3) the security policy in force.

3. The Printer MUST enforce its security and privacy policies based on the owner of the IPP object and the URI scheme and/or credentials supplied by the Client in the current request.

10. Changes since RFC 2910

The following changes have been made since the publication of RFC 2910:

- Added references to current IPP extension specifications.
- Added optional support for HTTP/2.
- Added collection attribute syntax from RFC 3382.
- Fixed typographical errors.
- Now reference TLS/1.2 and no longer mandate the TLS/1.0 MTI ciphersuites.
- Updated all references.
- Updated document organization to follow current style.
- Updated example ipp: URIs to follow guidelines in RFC 7472.
- Updated version compatibility for all versions of IPP.
- Updated HTTP Digest Authentication to optional for Clients.
- Removed references to (Experimental) IPP/1.0 and usage of http:/https: URLs.
11. References

11.1. Normative References


11.2. Informative References

[IANA-IPP] IANA, "Internet Printing Protocol (IPP) Registry",

[PWG5100.3]
Ocke, K. and T. Hastings, "Internet Printing Protocol (IPP): Production Printing Attributes - Set1",
Candidate Standard 5100.3-2001, February 2001,

RFC 1179, DOI 10.17487/rfc1179, August 1990,

BCP 188, RFC 7258, DOI 10.17487/rfc7258, May 2014,

RFC 7435, DOI 10.17487/rfc7435, December 2014,
## Appendix A. Protocol Examples

### A.1. Print-Job Request

The following is an example of a Print-Job request with "job-name", "copies", and "sides" specified. The "ipp-attribute-fidelity" attribute is set to 'true' so that the print request will fail if the "copies" or the "sides" attribute is not supported or their values are not supported.

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0002</td>
<td>Print-Job</td>
<td>operation-id</td>
</tr>
<tr>
<td>0x00000001</td>
<td>1</td>
<td>request-id</td>
</tr>
<tr>
<td>0x01</td>
<td>start operation-</td>
<td>operation-</td>
</tr>
<tr>
<td>0x47</td>
<td>charset type</td>
<td>attributes-charset</td>
</tr>
<tr>
<td>0x0005</td>
<td>attributes-charset</td>
<td>name</td>
</tr>
<tr>
<td>utf-8</td>
<td>UTF-8</td>
<td>value</td>
</tr>
<tr>
<td>0x48</td>
<td>natural-language</td>
<td>type</td>
</tr>
<tr>
<td>0x000b</td>
<td>attributes-natural-</td>
<td>name</td>
</tr>
<tr>
<td>printer-uri</td>
<td>printer-uri</td>
<td>name</td>
</tr>
<tr>
<td>0x002c</td>
<td>ipp://printer.example.com/ipp/print/pinetree</td>
<td>value</td>
</tr>
<tr>
<td>0x42</td>
<td>nameWithoutLanguage</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0005</td>
<td>job-name</td>
<td>name</td>
</tr>
<tr>
<td>0x0006</td>
<td>foobar</td>
<td>value-length</td>
</tr>
<tr>
<td>0x22</td>
<td>booleans</td>
<td>value</td>
</tr>
<tr>
<td>0x0016</td>
<td>ipp-attribute-fidelity</td>
<td>name</td>
</tr>
<tr>
<td>0x0001</td>
<td>true</td>
<td>value-length</td>
</tr>
<tr>
<td>0x02</td>
<td>start job-attributes</td>
<td>job-attributes-</td>
</tr>
</tbody>
</table>
A.2. Print-Job Response (Successful)

Here is an example of a successful Print-Job response to the previous Print-Job request. The Printer supported the "copies" and "sides" attributes and their supplied values. The status-code returned is ‘successful-ok’.

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0101</td>
<td>1.1</td>
<td>version-number</td>
</tr>
<tr>
<td>0x0000</td>
<td>successful-ok</td>
<td>status-code</td>
</tr>
<tr>
<td>0x000000001</td>
<td>1</td>
<td>request-id</td>
</tr>
<tr>
<td>0x01</td>
<td>start operation-</td>
<td>attributes-attributes-tag</td>
</tr>
<tr>
<td>0x47</td>
<td>charset type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0012</td>
<td>attributes-charset</td>
<td>name</td>
</tr>
<tr>
<td>0x0005</td>
<td>utf-8</td>
<td>value-length</td>
</tr>
<tr>
<td>0x48</td>
<td>natural-language type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x001b</td>
<td>attributes-natural-</td>
<td>name</td>
</tr>
<tr>
<td>0x0005</td>
<td>en-us</td>
<td>value-length</td>
</tr>
<tr>
<td>0x41</td>
<td>textWithoutLanguage type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x000e</td>
<td>status-message</td>
<td>name</td>
</tr>
<tr>
<td>0x000d</td>
<td>successful-ok</td>
<td>value-length</td>
</tr>
<tr>
<td>0x02</td>
<td>start job-attributes-tag</td>
<td>name</td>
</tr>
</tbody>
</table>
A.3. Print-Job Response (Failure)

Here is an example of an unsuccessful Print-Job response to the previous Print-Job request. It fails because, in this case, the Printer does not support the "sides" attribute and because the value '20' for the "copies" attribute is not supported. Therefore, no Job is created, and neither a "job-id" nor a "job-uri" operation attribute is returned. The error code returned is 'client-error-attributes-or-values-not-supported' (0x040b).

Octets        Symbolic Value        Protocol field

0x0101          1.1                   version-number
0x040b          client-error-attributes-or-values-not-supported
0x00000001      1                      request-id
0x01            start operation-attributes
0x47            charset type
0x0012          attributes-charset
0x0005          utf-8
0x48            natural-language type
0x001b
A.4. Print-Job Response (Success with Attributes Ignored)

Here is an example of a successful Print-Job response to a Print-Job request like the previous Print-Job request, except that the value of "ipp-attribute-fidelity" is 'false'. The print request succeeds, even though, in this case, the Printer supports neither the "sides" attribute nor the value '20' for the "copies" attribute. Therefore, a Job is created and both a "job-id" and a "job-uri" operation attribute are returned. The unsupported attributes are also returned in an Unsupported Attributes group. The error code returned is ‘successful-ok-ignored-or-substituted-attributes’ (0x0001).

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0101</td>
<td>1.1</td>
<td>version-number</td>
</tr>
<tr>
<td>0x0001</td>
<td>successful-ok-ignored-or-substituted-attributes</td>
<td>status-code</td>
</tr>
<tr>
<td>0x00000001</td>
<td>1</td>
<td>request-id</td>
</tr>
<tr>
<td>0x01</td>
<td>start operation-attributes</td>
<td>operation-attributes-tag</td>
</tr>
<tr>
<td>0x47</td>
<td>charset type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0012</td>
<td>attributes-charset</td>
<td>name-length</td>
</tr>
</tbody>
</table>

Sweet & McDonald Standards Track [Page 43]
utf-8
0x48
natural-language type
0x001b
name-length
attributes-natural-language
0x0005
en-us
0x41
textWithoutLanguage type
0x000e
name-length
status-message
0x0005
en-US
0x41
textWithoutLanguage type
0x000e
name-length
status-message
0x002f
value-length
successful-ok-ignored-or-
substituted-attributes
0x05
start unsupported-
attributes
0x21
integer type
0x0006
copies
0x0004
copies
0x00000014
20
value
0x10
unsupported (type)
0x0005
sides
0x0000
sides
0x0002
start job-attributes
0x23
enum type
0x0009
job-state
0x0004
job-state
0x0003
pending
0x03
end-of-attributes

Sweet & McDonald Standards Track [Page 44]
A.5. Print-URI Request

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

<table>
<thead>
<tr>
<th>Octettes</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0101</td>
<td>1.1</td>
<td>version-number</td>
</tr>
<tr>
<td>0x0003</td>
<td>Print-URI</td>
<td>operation-id</td>
</tr>
<tr>
<td>0x00000001</td>
<td>1</td>
<td>request-id</td>
</tr>
<tr>
<td>0x01</td>
<td>start operation-</td>
<td>attributes-</td>
</tr>
<tr>
<td></td>
<td>attributes-charset</td>
<td>attributes-</td>
</tr>
<tr>
<td>0x47</td>
<td>charset type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0012</td>
<td>attributes-charset</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0005</td>
<td>utf-8</td>
<td>value-length</td>
</tr>
<tr>
<td>0x48</td>
<td>natural-language</td>
<td>value-tag</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>0x001b</td>
<td>attributes-natural-language</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0005</td>
<td>en-us</td>
<td>value-length</td>
</tr>
<tr>
<td>0x45</td>
<td>uri type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0000b</td>
<td>printer-uri</td>
<td>name-length</td>
</tr>
<tr>
<td>0x002c</td>
<td>ipp://printer.example.com/ipp/printer-pinetree</td>
<td>value-length</td>
</tr>
<tr>
<td></td>
<td>print/pinetree</td>
<td></td>
</tr>
<tr>
<td>0x45</td>
<td>uri type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x000c</td>
<td>document-uri</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0019</td>
<td>ftp://foo.example.com/foo</td>
<td>value-length</td>
</tr>
<tr>
<td></td>
<td>ftp://foo.example.co/m/foo</td>
<td></td>
</tr>
<tr>
<td>0x42</td>
<td>nameWithoutLanguage</td>
<td>value-tag</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>0x0008</td>
<td>job-name</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0006</td>
<td>fooobar</td>
<td>value-length</td>
</tr>
<tr>
<td>0x02</td>
<td>start job-attributes</td>
<td>job-attributes-</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>0x21</td>
<td>integer type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0006</td>
<td>copies</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0004</td>
<td>copies</td>
<td>name-length</td>
</tr>
</tbody>
</table>
A.6. Create-Job Request

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

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0101</td>
<td>1.1</td>
<td>version-number</td>
</tr>
<tr>
<td>0x0005</td>
<td>Create-Job</td>
<td>operation-id</td>
</tr>
<tr>
<td>0x00000001</td>
<td>1</td>
<td>request-id</td>
</tr>
<tr>
<td>0x01</td>
<td>start operation-</td>
<td>operation-attributes-tag</td>
</tr>
<tr>
<td>0x47</td>
<td>charset type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0012</td>
<td>attributes-charset</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0005</td>
<td>utf-8</td>
<td>value-length</td>
</tr>
<tr>
<td>0x48</td>
<td>natural-language type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x001b</td>
<td>attributes-natural-language</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0005</td>
<td>en-us</td>
<td>value-length</td>
</tr>
<tr>
<td>0x45</td>
<td>uri type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x000b</td>
<td>printer-uri</td>
<td>name-length</td>
</tr>
<tr>
<td>0x002c</td>
<td>ipp://printer.example.com/ipp/printer-pinetree</td>
<td>value-length</td>
</tr>
<tr>
<td>0x03</td>
<td>end of attributes</td>
<td>end-of-attributes-tag</td>
</tr>
</tbody>
</table>

A.7. Create-Job Request with Collection Attributes

The following is an example of Create-Job request with the "media-col" collection attribute [PWG5100.3] with the value "media-size={x-dimension=21000 y-dimension=29700} media-type='stationery’":

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0101</td>
<td>1.1</td>
<td>version-number</td>
</tr>
<tr>
<td>0x0005</td>
<td>Create-Job</td>
<td>operation-id</td>
</tr>
<tr>
<td>0x00000001</td>
<td>1</td>
<td>request-id</td>
</tr>
</tbody>
</table>
0x01                           start operation-
operation-
attributes           attributes-tag

0x47                           charset type         value-tag
0x0012                                              name-length
attributes-charset             attributes-charset   name
0x0005                                              value-length
utf-8                                  UTF-8                value
type
0x001b                           attributes-natural-language
attributes-natural-language
name
0x0005                                              value-length
ten-us                          en-US                value
0x45                                              value-tag
uri type                           uri type             name-length
0x000b                                              value-length
printer-uri                          printer-uri          name
0x002c                                              value-length
ipp://printer.example.com/ipp/printer pinetree
print/pinetree
0x34                           begCollection        value-tag
0x0009                         9                    name-length
media-col                          media-col            name
0x0000                                              value-length
0x4a                                           memberAttrName   value-tag
0x0000                                              name-length
0x000a                                              value-length
media-size                          media-size           value (member-
name)
0x34                           begCollection        member-value-tag
0x0000                                              name-length
0x0000                                              member-value-
length
0x4a                                           memberAttrName   value-tag
0x0000                                              name-length
0x000b                                              value-length
x-dimension                        x-dimension          value (member-
name)
0x21                           integer            member-value-tag
0x0000                                              name-length
0x0004                                              member-value-
length
0x00005208                                      21000                member-value
0x4a                                           memberAttrName   value-tag
0x0000                                              name-length
0x000b                                              value-length
y-dimension                        y-dimension          value (member-
name)
A.8. Get-Jobs Request

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

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000007b</td>
<td>123</td>
<td>request-id</td>
</tr>
<tr>
<td>0x00000007b</td>
<td>start operation-</td>
<td>attributes</td>
</tr>
<tr>
<td>0x00012</td>
<td>attributes-charset</td>
<td>name</td>
</tr>
<tr>
<td>0x00005</td>
<td>utf-8</td>
<td>value-length</td>
</tr>
<tr>
<td>0x048</td>
<td>natural-language type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0001b</td>
<td>attributes-natural-</td>
<td>name-length</td>
</tr>
<tr>
<td></td>
<td>language name</td>
<td></td>
</tr>
<tr>
<td>0x00005</td>
<td>en-us</td>
<td>value-length</td>
</tr>
</tbody>
</table>

The following is an example of a Get-Jobs response from a previous request with three Jobs. The Printer returns no information about the second Job (because of security reasons):

<table>
<thead>
<tr>
<th>Octets</th>
<th>Symbolic Value</th>
<th>Protocol field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0101</td>
<td>1.1</td>
<td>version-number</td>
</tr>
<tr>
<td>0x0000</td>
<td>successful-ok</td>
<td>status-code</td>
</tr>
<tr>
<td>0x0000007b</td>
<td>123</td>
<td>request-id (echoed back)</td>
</tr>
<tr>
<td>0x01</td>
<td>start operation-attributes</td>
<td>operation-attributes-tag</td>
</tr>
<tr>
<td>0x47</td>
<td>charset type</td>
<td>value-tag</td>
</tr>
<tr>
<td>0x0012</td>
<td>attributes-charset</td>
<td>name-length</td>
</tr>
<tr>
<td>0x0005</td>
<td>value-length</td>
<td></td>
</tr>
<tr>
<td>utf-8</td>
<td>UTF-8</td>
<td>value</td>
</tr>
<tr>
<td>0x48</td>
<td>natural-language type</td>
<td>value-length</td>
</tr>
<tr>
<td>0x001b</td>
<td></td>
<td>name-length</td>
</tr>
</tbody>
</table>
attributes-natural-language
0x0005 en-us
0x041 textWithoutLanguage

0x000e status-message
0x000d successful-ok
0x02 start job-attributes
(1st object)

0x21 integer type
0x0006 job-id
0x0004 147
0x36 nameWithLanguage

0x0008 job-name
0x000c value-length
0x0005 sub-value-length

fr-ca fr-CA
0x0003 fou
0x02 start job-attributes
(2nd object)

0x02 start job-attributes
(3rd object)

0x21 integer type
0x0006 job-id
0x0004 148
0x36 nameWithLanguage

0x0008 job-name
0x0012 value-length
0x0005 sub-value-length

de-CH
de-CH
0x0009 sub-value-length

isch guet

0x03 end-of-attributes
end-of-attributes-tag
Acknowledgements

The authors would like to acknowledge the following individuals for their contributions to the original IPP/1.1 specifications:

Sylvan Butler, Roger deBry, Tom Hastings, Robert Herriot (the original editor of RFC 2910), Paul Moore, Kirk Ocke, Randy Turner, John Wenn, and Peter Zehler.

Authors’ Addresses

Michael Sweet
Apple Inc.
1 Infinite Loop
MS 111-HOMC
Cupertino, CA 95014
United States of America

Email: msweet@apple.com

Ira McDonald
High North, Inc.
PO Box 221
Grand Marais, MI 49839
United States of America

Phone: +1 906-494-2434
Email: blueroofmusic@gmail.com