              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.

Table of Contents

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:

    o Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing
       [RFC7230]

    o Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
       [RFC7231]

    o Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests
       [RFC7232]

    o Hypertext Transfer Protocol (HTTP/1.1): Caching [RFC7234]

    o Hypertext Transfer Protocol (HTTP/1.1): Authentication [RFC7235]

    o The 'Basic' HTTP Authentication Scheme [RFC7617]

    o HTTP Digest Access Authentication [RFC7616]

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

    o Hypertext Transfer Protocol Version 2 (HTTP/2) [RFC7540]

    o 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]

   ASCII: American Standard Code for Information Interchange [RFC20]

   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:

o   informally through pictures and description

o   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)           |
|                   or                      |   2 bytes  - required
|          status-code (response)           |
-------------------------------------------------
|                 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
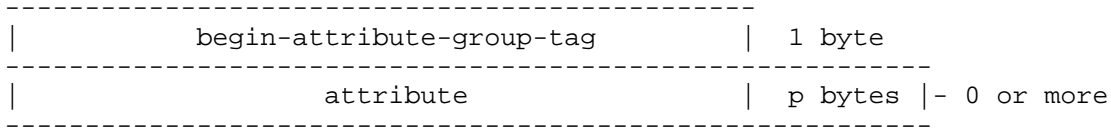
                  Figure 2: Attribute Group Encoding

   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
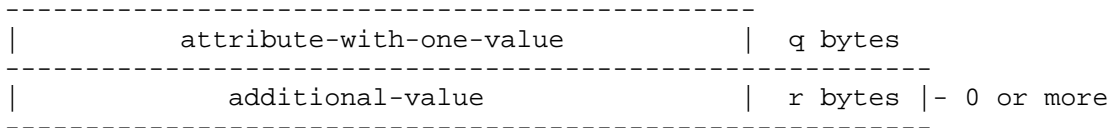
                     Figure 3: Attribute Encoding

   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:

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

   o 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".

   o The "name" field contains the textual name of the attribute, e.g.,
      "sides-supported".

   o 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'.

   o 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:

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

   o 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).

   o 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'.

   o 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:

```
   -------------------------------------------------
   |          value-tag (value is 0x34)          |   1 byte
   -------------------------------------------------
   |          name-length (value is u)           |   2 bytes
   -------------------------------------------------
   |                    name                     |   u bytes
   -------------------------------------------------
   |        value-length (value is 0x0000)       |   2 bytes
   ---------------------------------------------------------
   |                 member-attribute            |   q bytes |-0 or more
   ---------------------------------------------------------
   |          end-value-tag (value is 0x37)      |   1 byte
   -------------------------------------------------
   |      end-name-length (value is 0x0000)      |   2 bytes
   -------------------------------------------------
   |      end-value-length (value is 0x0000)     |   2 bytes
   -------------------------------------------------
```

                Figure 6: Collection Attribute Encoding

   Collection attribute is encoded with eight subfields:

   o The "value-tag" field specifies the start attribute syntax: 0x34
      for the attribute syntax 'begCollection'.

   o 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.

   o The "name" field contains the textual name of the attribute, e.g.,
      "media-col".

   o The "value-length" field specifies a length of 0x0000.

   o The "member-attribute" field contains member attributes encoded as
      defined in Section 3.1.7.

   o The "end-value-tag" field specifies the end attribute syntax: 0x37
      for the attribute syntax 'endCollection'.

   o The "end-name-length" field specifies a length of 0x0000.

   o The "end-value-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:

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

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

   o 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.

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

   o The "member-value-tag" field specifies the attribute syntax for
      the member attribute, e.g., 0x44 for the attribute syntax
      'keyword'.

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

   o 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'.

   o 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:

   | version-number | 2 bytes - required

   | operation-id (request) |
   | or | 2 bytes - required
   | status-code (response) |

   | request-id | 4 bytes - required

   | tag (delimiter-tag or value-tag) | 1 byte |
   --- |-0 or more
   | empty or rest of attribute | x bytes |

   | end-of-attributes-tag | 1 byte   - required

   | data | y bytes - optional

                 Figure 8: Encoding Based on Value Tags

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

   o "begin-attribute-group-tag": expect zero or more "attribute"
      fields

   o "value-tag": expect the remainder of an "attribute-with-one-value"
      or an "additional-value"

   o "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:

```
+---------------+-------------------------------------------------+
| Model Document| "begin-attribute-group-tag" field values        |
| Group         |                                                 |
+---------------+-------------------------------------------------+
| Operation     | "operations-attributes-tag"                     |
| Attributes    |                                                 |
+---------------+-------------------------------------------------+
| Job Template  | "job-attributes-tag"                            |
| Attributes    |                                                 |
+---------------+-------------------------------------------------+
| Job Object    | "job-attributes-tag"                            |
| Attributes    |                                                 |
+---------------+-------------------------------------------------+
| Unsupported   | "unsupported-attributes-tag"                    |
| Attributes    |                                                 |
+---------------+-------------------------------------------------+
| Requested     | (Get-Job-Attributes) "job-attributes-tag"       |
| Attributes    |                                                 |
+---------------+-------------------------------------------------+
| Requested     | (Get-Printer-Attributes)"printer-attributes-tag"|
| Attributes    |                                                 |
+---------------+-------------------------------------------------+
| Document      | in a special position at the end of the message |
| Content       | as described in Section 3.1.1.                  |
+---------------+-------------------------------------------------+
```

                        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:

   o delimiter tags: delimit major sections of the protocol, namely
      attribute groups and data

   o 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.

Tag Value (Hex)	Meaning
0x00	Reserved
0x01	"operation-attributes-tag"
0x02	"job-attributes-tag"
0x03	"end-of-attributes-tag"
0x04	"printer-attributes-tag"
0x05	"unsupported-attributes-tag"

                  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.

```
+-----------------+-------------+
| Tag Value (Hex) | Meaning     |
+-----------------+-------------+
| 0x10            | unsupported |
| 0x12            | unknown     |
| 0x13            | no-value    |
+-----------------+-------------+
```

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.

```
+---------------+-----------------------------------------------+
| Tag Value     | Meaning                                       |
| (Hex)         |                                               |
+---------------+-----------------------------------------------+
| 0x20          | Unassigned integer data type (see IANA IPP    |
|               | registry)                                     |
| 0x21          | integer                                       |
| 0x22          | boolean                                       |
| 0x23          | enum                                          |
| 0x24-0x2f     | Unassigned integer data types (see IANA IPP   |
|               | registry)                                     |
+---------------+-----------------------------------------------+
```

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.

```
+---------------+-----------------------------------------------+
| Tag Value     | Meaning                                       |
| (Hex)         |                                               |
+---------------+-----------------------------------------------+
| 0x30          | octetString with an unspecified format        |
| 0x31          | dateTime                                      |
| 0x32          | resolution                                    |
| 0x33          | rangeOfInteger                                |
| 0x34          | begCollection                                 |
| 0x35          | textWithLanguage                              |
| 0x36          | nameWithLanguage                              |
| 0x37          | endCollection                                 |
| 0x38-0x3f     | Unassigned octetString data types (see IANA IPP |
|               | registry)                                     |
+---------------+-----------------------------------------------+
```

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.

```
+---------------+---------------------------------------------------+
| Tag Value     | Meaning                                           |
| (Hex)         |                                                   |
+---------------+---------------------------------------------------+
| 0x40          | Unassigned character-string data type (see IANA   |
|               | IPP registry)                                     |
| 0x41          | textWithoutLanguage                               |
| 0x42          | nameWithoutLanguage                               |
| 0x43          | Unassigned character-string data type (see IANA   |
|               | IPP registry)                                     |
| 0x44          | keyword                                           |
| 0x45          | uri                                               |
| 0x46          | uriScheme                                         |
| 0x47          | charset                                           |
| 0x48          | naturalLanguage                                   |
| 0x49          | mimeMediaType                                     |
| 0x4a          | memberAttrName                                    |
| 0x4b-0x5f     | Unassigned character-string data types (see IANA  |
|               | IPP registry)                                     |
+---------------+---------------------------------------------------+
```

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
0x3fffffff 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.

Syntax of Attribute Value	Encoding
textWithoutLanguage, nameWithoutLanguage	LOCALIZED-STRING
textWithLanguage	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.
nameWithLanguage	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.
charset, naturalLanguage, mimeMediaType, keyword, uri, and uriScheme	US-ASCII-STRING
boolean	SIGNED-BYTE where 0x00 is 'false' and 0x01 is 'true'
integer and enum	a SIGNED-INTEGER

```
+--------------------+--------------------------------------------+
| dateTime           | OCTET-STRING consisting of eleven octets    |
|                    | whose contents are defined by               |
|                    | "DateAndTime" in RFC 2579 [RFC2579]         |
+--------------------+--------------------------------------------+
| resolution         | OCTET-STRING consisting of nine octets of   |
|                    | two SIGNED-INTEGERs followed by a SIGNED-   |
|                    | BYTE.  The first SIGNED-INTEGER contains    |
|                    | the value of cross-feed direction           |
|                    | resolution.  The second SIGNED-INTEGER      |
|                    | contains the value of feed direction        |
|                    | resolution.  The SIGNED-BYTE contains the   |
|                    | units value.                                |
+--------------------+--------------------------------------------+
| rangeOfInteger     | Eight octets consisting of two SIGNED-      |
|                    | INTEGERs.  The first SIGNED-INTEGER         |
|                    | contains the lower bound and the second     |
|                    | SIGNED-INTEGER contains the upper bound.    |
+--------------------+--------------------------------------------+
| 1setOf X           | Encoding according to the rules for an      |
|                    | attribute with more than one value.  Each   |
|                    | value X is encoded according to the rules   |
|                    | for encoding its type.                      |
+--------------------+--------------------------------------------+
| octetString        | OCTET-STRING                                |
+--------------------+--------------------------------------------+
| collection         | Encoding as defined in Section 3.1.6.       |
+--------------------+--------------------------------------------+
```

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.

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:

   o the target URI for the operation; and

   o 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:

   o Job attributes:

      * job-uri

      * job-printer-uri

   o Printer attributes:

      * printer-uri-supported

   o 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-URI) 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

Additional information:

    Deprecated alias names for this type: N/A
    Magic number(s): N/A
    File extension(s): N/A
    Macintosh file type code(s): N/A

Person & email address to contact for further information:

    ISTO PWG IPP Workgroup <ipp@pwg.org>

Intended usage: COMMON

Restrictions on usage: N/A

Author: ISTO PWG IPP Workgroup <ipp@pwg.org>

Change controller: ISTO PWG IPP Workgroup <ipp@pwg.org>

Provisional registration? (standards tree only): No

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:

   o understand any valid request whose major "version-number" is
      greater than 0; and

   o 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:

   o 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:

   o Added references to current IPP extension specifications.

   o Added optional support for HTTP/2.

   o Added collection attribute syntax from RFC 3382.

   o Fixed typographical errors.

   o Now reference TLS/1.2 and no longer mandate the TLS/1.0 MTI
      ciphersuites.

   o Updated all references.

   o Updated document organization to follow current style.

   o Updated example ipp: URIs to follow guidelines in RFC 7472.

   o Updated version compatibility for all versions of IPP.

   o Updated HTTP Digest Authentication to optional for Clients.

   o Removed references to (Experimental) IPP/1.0 and usage of
      http:/https: URLs.

11. References

11.1. Normative References

   [PWG5100.12]
             Sweet, M. and I. McDonald, "IPP Version 2.0, 2.1, and
             2.2", October 2015, <http://ftp.pwg.org/pub/pwg/standards/
             std-ipp20-20151030-5100.12.pdf>.

   [RFC20]   Cerf, V., "ASCII format for network interchange", STD 80,
             RFC 20, DOI 10.17487/RFC0020, October 1969,
             <http://www.rfc-editor.org/info/rfc20>.

   [RFC793]   Postel, J., "Transmission Control Protocol", STD 7,
             RFC 793, DOI 10.17487/RFC0793, September 1981,
             <http://www.rfc-editor.org/info/rfc793>.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
             Requirement Levels", BCP 14, RFC 2119,
             DOI 10.17487/RFC2119, March 1997,
             <http://www.rfc-editor.org/info/rfc2119>.

   [RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J.
             Schoenwaelder, Ed., "Textual Conventions for SMIv2",
             STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999,
             <http://www.rfc-editor.org/info/rfc2579>.

   [RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within
             HTTP/1.1", RFC 2817, DOI 10.17487/RFC2817, May 2000,
             <http://www.rfc-editor.org/info/rfc2817>.

   [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818,
             DOI 10.17487/RFC2818, May 2000,
             <http://www.rfc-editor.org/info/rfc2818>.

   [RFC2978] Freed, N. and J. Postel, "IANA Charset Registration
             Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978,
             October 2000, <http://www.rfc-editor.org/info/rfc2978>.

   [RFC3510] Herriot, R. and I. McDonald, "Internet Printing
             Protocol/1.1: IPP URL Scheme", RFC 3510,
             DOI 10.17487/RFC3510, April 2003,
             <http://www.rfc-editor.org/info/rfc3510>.

   [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
             10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November
             2003, <http://www.rfc-editor.org/info/rfc3629>.

   [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
              Resource Identifier (URI): Generic Syntax", STD 66,
              RFC 3986, DOI 10.17487/RFC3986, January 2005,
              <http://www.rfc-editor.org/info/rfc3986>.

   [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", STD 68, RFC 5234,
              DOI 10.17487/RFC5234, January 2008,
              <http://www.rfc-editor.org/info/rfc5234>.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246,
              DOI 10.17487/RFC5246, August 2008,
              <http://www.rfc-editor.org/info/rfc5246>.

   [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Message Syntax and Routing",
              RFC 7230, DOI 10.17487/RFC7230, June 2014,
              <http://www.rfc-editor.org/info/rfc7230>.

   [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
              DOI 10.17487/RFC7231, June 2014,
              <http://www.rfc-editor.org/info/rfc7231>.

   [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Conditional Requests", RFC 7232,
              DOI 10.17487/RFC7232, June 2014,
              <http://www.rfc-editor.org/info/rfc7232>.

   [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke,
              Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching",
              RFC 7234, DOI 10.17487/RFC7234, June 2014,
              <http://www.rfc-editor.org/info/rfc7234>.

   [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Authentication", RFC 7235,
              DOI 10.17487/RFC7235, June 2014,
              <http://www.rfc-editor.org/info/rfc7235>.

   [RFC7472] McDonald, I. and M. Sweet, "Internet Printing Protocol
              (IPP) over HTTPS Transport Binding and the 'ipps' URI
              Scheme", RFC 7472, DOI 10.17487/RFC7472, March 2015,
              <http://www.rfc-editor.org/info/rfc7472>.

   [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext
              Transfer Protocol Version 2 (HTTP/2)", RFC 7540,
              DOI 10.17487/RFC7540, May 2015,
              <http://www.rfc-editor.org/info/rfc7540>.

   [RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for
              HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015,
              <http://www.rfc-editor.org/info/rfc7541>.

   [RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP
              Digest Access Authentication", RFC 7616,
              DOI 10.17487/RFC7616, September 2015,
              <http://www.rfc-editor.org/info/rfc7616>.

   [RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme",
              RFC 7617, DOI 10.17487/RFC7617, September 2015,
              <http://www.rfc-editor.org/info/rfc7617>.

   [RFC8011] Sweet, M. and I. McDonald, "Internet Printing
              Protocol/1.1: Model and Semantics", RFC 8011,
              DOI 10.17487/RFC8011, January 2017,
              <http://www.rfc-editor.org/info/rfc8011>.

11.2. Informative References

   [IANA-IPP] IANA, "Internet Printing Protocol (IPP) Registry",
              <http://www.iana.org/assignments/ipp-registrations/>.

   [PWG5100.3]
              Ocke, K. and T. Hastings, "Internet Printing Protocol
              (IPP): Production Printing Attributes - Set1", Candidate
              Standard 5100.3-2001, February 2001,
              <http://ftp.pwg.org/pub/pwg/candidates/
              cs-ippprodprint10-20010212-5100.3.pdf>.

   [RFC1179] McLaughlin, L., "Line printer daemon protocol", RFC 1179,
              DOI 10.17487/RFC1179, August 1990,
              <http://www.rfc-editor.org/info/rfc1179>.

   [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an
              Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May
              2014, <http://www.rfc-editor.org/info/rfc7258>.

   [RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection
              Most of the Time", RFC 7435, DOI 10.17487/RFC7435,
              December 2014, <http://www.rfc-editor.org/info/rfc7435>.

   [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre,
              "Recommendations for Secure Use of Transport Layer
              Security (TLS) and Datagram Transport Layer Security
              (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May
              2015, <http://www.rfc-editor.org/info/rfc7525>.

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.

Octets	Symbolic Value	Protocol field
0x0101	1.1	version-number
0x0002	Print-Job	operation-id
0x00000001	1	request-id
0x01	start operation-attributes	operation-attributes-tag
0x47	charset type	value-tag
0x0012		name-length
attributes-charset	attributes-charset	name
0x0005		value-length
utf-8	UTF-8	value
0x48	natural-language type	value-tag
0x001b		name-length
attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value
0x45	uri type	value-tag
0x000b		name-length
printer-uri	printer-uri	name
0x002c		value-length
ipp://printer.example.com/ipp/print/pinetree	printer pinetree	value
0x42	nameWithoutLanguage type	value-tag
0x0008		name-length
job-name	job-name	name
0x0006		value-length
foobar	foobar	value
0x22	boolean type	value-tag
0x0016		name-length
ipp-attribute-fidelity	ipp-attribute-fidelity	name
0x0001		value-length
0x01	true	value
0x02	start job-attributes	job-attributes-

```
                                            tag
   0x21                         integer type        value-tag
   0x0006                                           name-length
   copies                       copies              name
   0x0004                                           value-length
   0x00000014                   20                  value
   0x44                         keyword type        value-tag
   0x0005                                           name-length
   sides                        sides               name
   0x0013                                           value-length
   two-sided-long-edge          two-sided-long-edge value
   0x03                         end-of-attributes   end-of-
                                                    attributes-tag
   %!PDF...                     <PDF Document>      data
```

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'.

```
   Octets                       Symbolic Value      Protocol field

   0x0101                       1.1                 version-number
   0x0000                       successful-ok       status-code
   0x00000001                   1                   request-id
   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
   0x48                         natural-language    value-tag
                                type
   0x001b                                           name-length
   attributes-natural-language  attributes-         name
                                natural-language
   0x0005                                           value-length
   en-us                        en-US               value
   0x41                         textWithoutLanguag  value-tag
                                e type
   0x000e                                           name-length
   status-message               status-message      name
   0x000d                                           value-length
   successful-ok                successful-ok       value
   0x02                         start job-          job-attributes-
```

Octets	Symbolic Value	Protocol field
	attributes	tag
0x21	integer	value-tag
0x0006		name-length
job-id	job-id	name
0x0004		value-length
147	147	value
0x45	uri type	value-tag
0x0007		name-length
job-uri	job-uri	name
0x0030		value-length
ipp://printer.example.com/ipp/pr int/pinetree/147	job 147 on pinetree	value
0x23	enum type	value-tag
0x0009		name-length
job-state	job-state	name
0x0004		value-length
0x0003	pending	value
0x03	end-of-attributes	end-of-attributes-tag

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	status-code
0x00000001	1	request-id
0x01	start operation-attributes	operation-attributes tag
0x47	charset type	value-tag
0x0012		name-length
attributes-charset	attributes-charset	name
0x0005		value-length
utf-8	UTF-8	value
0x48	natural-language type	value-tag
0x001b		name-length

Octets	Symbolic Value	Protocol field
attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value
0x41	textWithoutLanguage type	value-tag
0x000e		name-length
status-message	status-message	name
0x002f		value-length
client-error-attributes-or-values-not-supported	client-error-attributes-or-values-not-supported	value
0x05	start unsupported-attributes	unsupported-attributes tag
0x21	integer type	value-tag
0x0006		name-length
copies	copies	name
0x0004		value-length
0x00000014	20	value
0x10	unsupported (type)	value-tag
0x0005		name-length
sides	sides	name
0x0000		value-length
0x03	end-of-attributes	end-of-attributes-tag

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).

Octets	Symbolic Value	Protocol field
0x0101	1.1	version-number
0x0001	successful-ok-ignored-or-substituted-attributes	status-code
0x00000001	1	request-id
0x01	start operation-attributes	operation-attributes-tag
0x47	charset type	value-tag
0x0012		name-length
attributes-charset	attributes-charset	name

0x0005		value-length
utf-8	UTF-8	value
0x48	natural-language type	value-tag
0x001b		name-length
attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value
0x41	textWithoutLanguage type	value-tag
0x000e		name-length
status-message	status-message	name
0x002f		value-length
successful-ok-ignored-or-substituted-attributes	successful-ok-ignored-or-substituted-attributes	value
0x05	start unsupported-attributes	unsupported-attributes tag
0x21	integer type	value-tag
0x0006		name-length
copies	copies	name
0x0004		value-length
0x00000014	20	value
0x10	unsupported (type)	value-tag
0x0005		name-length
sides	sides	name
0x0000		value-length
0x02	start job-attributes	job-attributes-tag
0x21	integer	value-tag
0x0006		name-length
job-id	job-id	name
0x0004		value-length
147	147	value
0x45	uri type	value-tag
0x0007		name-length
job-uri	job-uri	name
0x0030		value-length
ipp://printer.example.com/ipp/print/pinetree/147	job 147 on pinetree	value
0x23	enum type	value-tag
0x0009		name-length
job-state	job-state	name
0x0004		value-length
0x0003	pending	value
0x03	end-of-attributes	end-of-attributes-tag

A.5. Print-URI Request

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

Octets	Symbolic Value	Protocol field
0x0101	1.1	version-number
0x0003	Print-URI	operation-id
0x00000001	1	request-id
0x01	start operation-attributes	operation-attributes-tag
0x47	charset type	value-tag
0x0012		name-length
attributes-charset	attributes-charset	name
0x0005		value-length
utf-8	UTF-8	value
0x48	natural-language type	value-tag
0x001b		name-length
attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value
0x45	uri type	value-tag
0x000b		name-length
printer-uri	printer-uri	name
0x002c		value-length
ipp://printer.example.com/ipp/print/pinetree	printer pinetree	value
0x45	uri type	value-tag
0x000c		name-length
document-uri	document-uri	name
0x0019		value-length
ftp://foo.example.com/foo	ftp://foo.example.com/foo	value
0x42	nameWithoutLanguage type	value-tag
0x0008		name-length
job-name	job-name	name
0x0006		value-length
foobar	foobar	value
0x02	start job-attributes	job-attributes-tag
0x21	integer type	value-tag
0x0006		name-length
copies	copies	name
0x0004		value-length

```
   0x00000001                    1                     value
   0x03                          end-of-attributes     end-of-
                                                       attributes-tag
```

A.6. Create-Job Request

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

```
   Octets                        Symbolic Value        Protocol field

   0x0101                        1.1                   version-number
   0x0005                        Create-Job            operation-id
   0x00000001                    1                     request-id
   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
   0x48                          natural-language      value-tag
                                 type
   0x001b                                              name-length
   attributes-natural-language   attributes-natural-   name
                                 language
   0x0005                                              value-length
   en-us                         en-US                 value
   0x45                          uri type              value-tag
   0x000b                                              name-length
   printer-uri                   printer-uri           name
   0x002c                                              value-length
   ipp://printer.example.com/ipp/ printer pinetree     value
   print/pinetree
   0x03                          end-of-attributes     end-of-
                                                       attributes-tag
```

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'":

```
   Octets                        Symbolic Value        Protocol field

   0x0101                        1.1                   version-number
   0x0005                        Create-Job            operation-id
   0x00000001                    1                     request-id
```

0x01	start operation-attributes	operation-attributes-tag
0x47	charset type	value-tag
0x0012		name-length
attributes-charset	attributes-charset	name
0x0005		value-length
utf-8	UTF-8	value
0x48	natural-language type	value-tag
0x001b		name-length
attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value
0x45	uri type	value-tag
0x000b		name-length
printer-uri	printer-uri	name
0x002c		value-length
ipp://printer.example.com/ipp/print/pinetree	printer pinetree	value
0x34	begCollection	value-tag
0x0009	9	name-length
media-col	media-col	name
0x0000	0	value-length
0x4a	memberAttrName	value-tag
0x0000	0	name-length
0x000a	10	value-length
media-size	media-size	value (member-name)
0x34	begCollection	member-value-tag
0x0000	0	name-length
0x0000	0	member-value-length
0x4a	memberAttrName	value-tag
0x0000	0	name-length
0x000b	11	value-length
x-dimension	x-dimension	value (member-name)
0x21	integer	member-value-tag
0x0000	0	name-length
0x0004	4	member-value-length
0x00005208	21000	member-value
0x4a	memberAttrName	value-tag
0x0000	0	name-length
0x000b	11	value-length
y-dimension	y-dimension	value (member-name)

0x21	integer	member-value-tag
0x0000	0	name-length
0x0004	4	member-value-length
0x00007404	29700	member-value
0x37	endCollection	end-value-tag
0x0000	0	end-name-length
0x0000	0	end-value-length
0x4a	memberAttrName	value-tag
0x0000	0	name-length
0x000a	10	value-length
media-type	media-type	value (member-name)
0x44	keyword	member-value-tag
0x0000	0	name-length
0x000a	10	member-value-length
stationery	stationery	member-value
0x37	endCollection	end-value-tag
0x0000	0	end-name-length
0x0000	0	end-value-length
0x03	end-of-attributes	end-of-attributes-tag

A.8. Get-Jobs Request

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

Octets	Symbolic Value	Protocol field
0x0101	1.1	version-number
0x000a	Get-Jobs	operation-id
0x0000007b	123	request-id
0x01	start operation-attributes	operation-attributes-tag
0x47	charset type	value-tag
0x0012		name-length
attributes-charset	attributes-charset	name
0x0005		value-length
utf-8	UTF-8	value
0x48	natural-language type	value-tag
0x001b		name-length
attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value

```
0x45                          uri type              value-tag
0x000b                                              name-length
printer-uri                   printer-uri           name
0x002c                                              value-length
ipp://printer.example.com/ipp/ printer pinetree     value
print/pinetree
0x21                          integer type          value-tag
0x0005                                              name-length
limit                         limit                 name
0x0004                                              value-length
0x00000032                    50                    value
0x44                          keyword type          value-tag
0x0014                                              name-length
requested-attributes          requested-attributes  name
0x0006                                              value-length
job-id                        job-id                value
0x44                          keyword type          value-tag
0x0000                        additional value      name-length
0x0008                                              value-length
job-name                      job-name              value
0x44                          keyword type          value-tag
0x0000                        additional value      name-length
0x000f                                              value-length
document-format               document-format       value
0x03                          end-of-attributes     end-of-
                                                    attributes-tag
```

A.9. Get-Jobs Response

   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):

```
Octets                   Symbolic Value         Protocol field

0x0101                   1.1                    version-number
0x0000                   successful-ok          status-code
0x0000007b               123                    request-id (echoed
                                                back)
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
0x48                     natural-language type  value-tag
0x001b                                          name-length
```

attributes-natural-language	attributes-natural-language	name
0x0005		value-length
en-us	en-US	value
0x41	textWithoutLanguage type	value-tag
0x000e		name-length
status-message	status-message	name
0x000d		value-length
successful-ok	successful-ok	value
0x02	start job-attributes (1st object)	job-attributes-tag
0x21	integer type	value-tag
0x0006		name-length
job-id	job-id	name
0x0004		value-length
147	147	value
0x36	nameWithLanguage	value-tag
0x0008		name-length
job-name	job-name	name
0x000c		value-length
0x0005		sub-value-length
fr-ca	fr-CA	value
0x0003		sub-value-length
fou	fou	name
0x02	start job-attributes (2nd object)	job-attributes-tag
0x02	start job-attributes (3rd object)	job-attributes-tag
0x21	integer type	value-tag
0x0006		name-length
job-id	job-id	name
0x0004		value-length
148	149	value
0x36	nameWithLanguage	value-tag
0x0008		name-length
job-name	job-name	name
0x0012		value-length
0x0005		sub-value-length
de-CH	de-CH	value
0x0009		sub-value-length
isch guet	isch guet	name
0x03	end-of-attributes	end-of-attributes-tag

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