Robert Herriot (editor)
Sun Microsystems
Sylvan Butler
Hewlett-Packard
Paul Moore
Microsoft.
Randy Turner
Sharp Labs
July 30~~14~~, 1997

Internet Printing Protocol/1.0: Protocol Specification

draft-ietf-ipp-protocol-01~~0~~.txt

Status of this Memo

Abstract

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

The full set of IPP documents includes:

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

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

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

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/1.1 request or response. RFC 2068 [27] describes HTTP/1.1. 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 document "Internet Printing Protocol/1.0: Model and Semantics" [21] defines the semantics of such a message body and the supported values. This document specifies the encoding of an IPP operation. The aforementioned document [21] is henceforth referred to as the "IPP model document"

2. Conformance Terminology

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 RFC 2119 [25].

3. Encoding of the Operation Layer

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

The encoding consists of octet as the most primitive type. There are several types built from octets, but two important ~~primitive~~ types are integers and characters, on which most ~~all~~ other data types are built. Every character in this encoding SHALL be a member of the UCS-2 coded character set and SHALL be encoded using UTF-8 which uses 1 to 3 octets per character. Every integer in this encoding SHALL be encoded ~~in binary~~ 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~~bytes~~ for an integer SHALL ~~a power of 2, that is,~~ be 1, 2 or 4, depending on usage in the protocol. Such one-octet integers, henceforth called SIGNED-BYTE, are used for the version and tag fields. Such two-byte integers, henceforth called SIGNED-SHORT are used for the operation, status-code and length fields. Four byte integers, henceforth called SIGNED-INTEGER, are used for values fields.

The following two sections present the operation layer in two ways

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

3.1 Picture of the Encoding

The encoding for an operation request or response consists of:

```
-----------------------------------------------
|                  version                    |   2 bytes  - required
-----------------------------------------------
|operation (request) or status-code (response)|   2 bytes  - required
-----------------------------------------------------------
|               parameter-tag                 |   1 byte  |
-----------------------------------------------           |- optional
|             parameter-sequence              |   m bytes |
-----------------------------------------------------------
|               attribute-tag                 |   1 byte  |
-----------------------------------------------           |- 0 or more
|             attribute-sequence              |   n bytes |
-----------------------------------------------------------
|                 data-tag                    |   1 byte   - required
-----------------------------------------------
|                   data                      |   q bytes  - optional
-----------------------------------------------
```

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

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

```
-----------------------------------------------
|             compound-parameter              |   r bytes - 0 or more
-----------------------------------------------
```

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

```
-----------------------------------------------
|             compound-attribute              |   s bytes - 0 or more
-----------------------------------------------
```

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

Each parameter or attribute consists of:

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

An additional value consists of:

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

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

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

```
---------------------------------------------
|                   version                  |   2 bytes  - required
---------------------------------------------
|operation (request) or status-code (response)|   2 bytes  - required
-----------------------------------------------------------
|        tag (delimiter-tag or value-tag)    |   1 byte  |
---------------------------------------------            |- 0 or more
|      empty or rest of parameter/attribute  |   x bytes |
-----------------------------------------------------------
|                  data-tag                  |   2 bytes  - required
---------------------------------------------
|                    data                    |   y bytes  - optional
---------------------------------------------
```

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

- parameters
- attributes
- data
- the remainder of a single parameter or attribute where the tag specifies the type of the value.

3.2 Syntax of Encoding

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

```
ipp-message = ipp-request / ipp-response
ipp-request = version operation [parameter-tag parameter-sequence ]
        *(attribute-tag  attribute-sequence) data-tag data
ipp-response = version status-code [parameter-tag parameter-sequence ]
        *(attribute-tag attribute-sequence)  data-tag  data

version = major-version minor-version
major-version = SIGNED-BYTE  ; initially %d1
minor-version = SIGNED-BYTE  ; initially %d0

operation = SIGNED-SHORT  ; mapping from model defined below
```

status-code = SIGNED-SHORT ; mapping from model defined below

parameter-sequence = *compound-parameter
attribute-sequence = *compound-attribute
compound-parameter = parameter *additional-values
compound-attribute = attribute *additional-values

parameter = value-tag name-length name value-length value
attribute = value-tag name-length name value-length value
additional-values = 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
parameter-tag = %x01 ; tag of 1
attribute-tag = %x02 ; tag of 2
data-tag = %x03 ; tag of 3
value-tag = %x10-~~..%x~~FF

SIGNED-BYTE = BYTE~~%x00..%xFF~~
SIGNED-SHORT = 2BYTE~~%x00..%xFF %x00..%xFF~~
DIGIT = %x30-39 ; "0" to ~~..~~"9"
LALPHA = %x61-7A ; "a" to ~~..~~"z"
BYTE = %x00-~~..%x~~FF
OCTET-STRING = *BYTE

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

3.3 Version

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

3.4 Mapping of Operations

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

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

3.5 Mapping of Status-code

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

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 SHALL NOT contain an IPP message-body, and thus no IPP status-code is returned.

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

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

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

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

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

3.6 Tags

There are two kinds of tags:

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

3.6.1 Delimiter Tags

The following table specifies the values for the delimiter tags:

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

3.6.2 Value Tags

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

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

Tag Value (Hex)	**Meaning**
0x10	unsupported
0x11	default
0x12	no-value
0x13~~2~~-0x1F	reserved for future “out-of-band” values.

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

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

Tag Value (Hex)	**Meaning**
0x20	reserved

Tag Value (Hex)	Meaning
0x21	integer ~~(up to 4 bytes)~~
0x22	boolean
0x23	enum~~seconds~~
~~0x24~~	~~milliseconds~~
~~0x25~~	~~enum~~
0x24~~6~~-0x2~~3~~F	reserved for future integer types

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

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

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

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

Tag Value (Hex)	Meaning
0x40	reserved
0x41	text
0x42	name
0x43	language~~filename~~
0x44	keyword
0x45	uri
0x46	uriScheme
~~0x47~~	~~dateTime~~
0x47~~8~~-0x5~~7~~F	reserved for future character string types

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

The values 0x6~~8~~0-0xFF are reserved for future types. There are no values allocated for private extensions. A new type must be registered via the type 2 process.

Issue: should this be a type 1 process.

3.7 Name-Lengths

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

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

3.8 Mapping of Parameter Names

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

- "printer-uri": The printer-uri of each printer object operation in the IPP model document SHALL be specified both as a parameter named "printer-uri" in the operation layer ,and outside of the operation layer as the request-URI on the Request-Line at the HTTP level,.
- "job-uri": The job -uri of each job object operation in the IPP model document SHALL be specified both as a parameter named "job -uri" in the operation layer ,and outside of the operation layer as the request-URI on the Request-Line at the HTTP level,.
- ~~"request-URI": The request-URI of each operation in the IPP model document SHALL be specified outside of the operation layer as the request-URI on the Request-Line at the HTTP level, and SHALL not be specified within the operation layer.~~
- "document-content": The parameter named "document-content" in the IPP model document SHALL become the "data" in the operation layer.
- "status-code": The parameter named "status-code" in the IPP model document SHALL become the "status-code" field in the operation layer response.

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

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

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

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

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

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

See Section 10 "Appendix B~~C~~: Mapping of Each Operation in the Encoding" for table showing the application of the rules above.

3.9 Value Lengths

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

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

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

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.

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

3.10 Mapping of Attribute and Parameter Values

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

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

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

3.11 Data

The data part SHALL include any data required by the operation

4. Encoding of Transport Layer

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

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

- the URI of the target job or printer operation
- 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.
- the client's language, the character-set and the transport encoding.

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

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

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

- the "header" column contains the name of a header
- the "request/client" column indicates whether a client sends the header.
- the "request/server" column indicates whether a server supports the header when received.
- the "response/server" column indicates whether a server sends the header.
- the "response /client" column indicates whether a client supports the header when received.
- the "values and conditions" column specifies the allowed header values and the conditions for the header to be present in a request/response.

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

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

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

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

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

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

4.1 General Headers

The following is a table for the general headers.

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

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

4.2 Request Headers

The following is a table for the request headers.

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

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

4.3 Response Headers

The following is a table for the request headers.

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

4.4 Entity Headers

The following is a table for the entity headers.

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

5. Security Considerations

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

For secure communication (privacy in particular), IPP should be run using a secure communications channel. Both Transport Layer Security - TLS (draft-ietf-tls-protocol-03) and IPSec (RFC 1825) provide necessary features for security. It is possible to combine the two techniques, HTTP 1.1 client authentication (either basic or digest) with secure communications channel (either TLS or IPSec). Together the two are more secure than client authentication and they perform user authentication.

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

ISSUE: how does each security mechanism supply the job-originating-user and job-originating-host values?

6. References

[1] Smith, R., Wright, F., Hastings, T., Zilles, S., and Gyllenskog, J., "Printer MIB", RFC 1759, March 1995.

[2] Berners-Lee, T, Fielding, R., and Nielsen, H., "Hypertext Transfer Protocol - HTTP/1.0", RFC 1945, August 1995.

[3] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", RFC 822, August 1982.

[4] Postel, J., "Instructions to RFC Authors", RFC 1543, October 1993.

[5] ISO/IEC 10175 Document Printing Application (DPA), June 1996.

[6] Herriot, R. (editor), X/Open A Printing System Interoperability Specification (PSIS), August 1995.

[7] Kirk, M. (editor), POSIX System Administration - Part 4: Printing Interfaces, POSIX 1387.4 D8, 1994.

[8] Borenstein, N., and Freed, N., "MIME (Multi-purpose Internet Mail Extensions) Part One: Mechanism for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September, 1993.

[9] Braden, S., "Requirements for Internet Hosts - Application and Support", RFC 1123, October, 1989,

[10] McLaughlin, L. III, (editor), "Line Printer Daemon Protocol" RFC 1179, August 1990.

[11] Berners-Lee, T., Masinter, L., McCahill, M. , "Uniform Resource Locators (URL)", RFC 1738, December, 1994.

[20] Wright, F. D., "Requirements for an Internet Printing Protocol:~~ Requirements~~"

[21] Isaacson, S, deBry, R, Hasting, T, Herriot, R, Powell, P, "Internet Printing Protocol/1.0: Model and Semantics"

[22] Internet Printing Protocol/1.0: Security

[23] Herriot, R, Butler, S, Moore, P, Turner, R, "Internet Printing Protocol/1.0: Protocol Specification" (This document)

[24] Carter, K, Isaacson, S, "Internet Printing Protocol/1.0: Directory Schema"

[25] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119 , March 1997

[26] H. Alvestrand, " Tags for the Identification of Languages", RFC 1766, March 1995.

[27] R Fielding, et al, "Hypertext Transfer Protocol – HTTP/1.1" RFC 2068, January 1997

[28] J. Case, et al. "Textual Conventions for Version 2 of the Simple Network Managment Protocol (SNMPv2)", RFC 1903, January 1996. ~~Marcus Kuhn, "International Standard Date and Time Notation", ISO 8601, http://www.ft.uni-erlangen.de/~mskuhn/iso-time.html~~

430 [29] D. Crocker et al., “Augmented BNF for Syntax Specifications: ABNF”, draft-ietf-drums-abnf-03~~2~~.txt.

431 [30] R. Turner, “Printer MIB”, draft-ietf-printmib-mib-info-02.txt, July 12, 1997.

432 # 7. Author's Address

433

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

Phone: 650~~415~~-786-8995
Fax: 650~~415~~-786-7077
Email: robert.herriot@eng.sun.com

Paul Moore
Microsoft
One Microsoft Way
Redmond, WA 98053

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

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

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

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

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

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

434

435 # 8. Other Participants:

Chuck Adams - Tektronix
Ron Bergman - Data Products
Keith Carter - IBM
Angelo Caruso - Xerox
Jeff Copeland - QMS
Roger Debry - IBM
Lee Farrell - Canon
Sue Gleeson - Digital
Charles Gordon - Osicom
Brian Grimshaw - Apple
Jerry Hadsell - IBM
Richard Hart - Digital
Tom Hastings - Xerox
Stephen Holmstead
Zhi-Hong Huang - Zenographics
Scott Isaacson - Novell
Rich Lomicka - Digital
David Kellerman - Northlake Software
Harry Lewis - IBM
Tony Liao - Vivid Image
David Manchala - Xerox
Carl-Uno Manros - Xerox
Jay Martin - Underscore
Larry Masinter - Xerox
Bob Pentecost - Hewlett-Packard
Patrick Powell - SDSU
Jeff Rackowitz - Intermec
Xavier Riley - Xerox
Gary Roberts - Ricoh
Stuart Rowley - Kyocera
Richard Schneider - Epson
Shigern Ueda - Canon
Bob Von Andel - Allegro Software
William Wagner - Digital Products
Jasper Wong - Xionics
Don Wright - Lexmark

Robert Kline - TrueSpectra
Dave Kuntz - Hewlett-Packard
Takami Kurono - Brother
Rich Landau - Digital
Greg LeClair - Epson
Rick Yardumian - Xerox
Lloyd Young - Lexmark
Peter Zehler - Xerox
Frank Zhao - Panasonic
Steve Zilles - Adobe

436 9. Appendix A: Protocol Examples

437 9.1 Print-Job Request

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

Octets	**Symbolic Value**	**Protocol field**
0x0100	1.0	version
0x0002	PrintJob	operation
0x02	start attributes	attribute tag
0x42	name type	value-tag
0x0008		name-length
job-name	job-name	name
0x0006		value-length
foobar	foobar	value
0x21	integer type	value-tag
0x0005		name-length
copies	copies	name
0x0004~~1~~		value-length
0x00000014	20	value
0x44	keyword type	value-tag
0x0005		name-length
sides~~copies~~	sides	name
0x0001		value-length
two-sided-long-edge	two-sided-long-edge	value
0x03	start-data	data-tag
%!PS...	<PostScript>	data

439 9.2 Print-Job Response (successful)

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

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

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

9.3 Print-Job Response (failure)

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

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x0400	client-error-bad-request	status-code
0x01	start parameters	parameter tag
0x41	text type	value-tag
0x000E		name-length
status-message	status-message	name
0x000D		value-length
bad-request	bad-request	value
0x02	start attributes	attribute tag
0x21	integer type	value-tag
0x0005		name-length
copies	copies	name
0x0004~~1~~		value-length
0x00000014	20	value
0x10	unsupported (type)	value-tag
0x0005		name-length
sides	sides	name
0x0000		value-length
0x03	start-data	data-tag

9.4 Print-URI Request

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

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

Octets	Symbolic Value	Protocol field
0x42	name type	value-tag
0x0008		name-length
job-name	job-name	name
0x0006		value-length
foobar	foobar	value
0x21	integer type	value-tag
0x0005		name-length
copies	copies	name
0x0004~~1~~		value-length
0x00000001	1	value
0x03	start-data	data-tag
%!PS...	<PostScript>	data

9.5 Create-Job Request

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

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

9.6 Get-Jobs Request

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

Octets	Symbolic Value	Protocol field
0x0100	1.0	version
0x000A	Get-Jobs	operation
0x01	start-parameters	parameter-tag
0x21	integer type	value-tag
0x0005		name-length
limit	limit	name
0x0004~~1~~		value-length
0x00000032	50	value
0x44	keyword type	value-tag
0x0014		name-length
requested-attributes	requested-attributes	name
0x0007		value-length
job-uri	job-uri	value
0x44	keyword type	value-tag
0x0000	additional value	name-length
0x0008		value-length
job-name	job-name	value
0x03	start-data	data-tag

9.7 Get-Jobs Response

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

Octets	Symbolic Value	Protocol field

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

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

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

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

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

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

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

~~**HTTP header or other concept**~~	~~**IPP header name**~~	~~**Syntax Type of header**~~

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

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

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

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

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

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

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

9.8 Chunking of Data

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

The following is the informal picture of chunked-data:

```
-----------------------------------------------------------
|              data-segment-length               |  2 bytes |
-----------------------------------------------             |- 0 or more
|                 data-segment                   |  n bytes |
-----------------------------------------------------------
|              0x0000 (end-of-data)              |  2 bytes   - required
-----------------------------------------------
```

The syntax for the chunked-data using ABNF is:

chunked-data = *data-chunk end-of-data
data-chunk = data-segment-length data-segment

data-segment-length = SIGNED-SHORT ; number of octets of the data in binary
data-segment = OCTET-STRING

end-of-data = %x00.00

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

9.8 Revised Picture for the Operation Layer

The encoding for an operation request or response consists of:

```
-----------------------------------------------
|                  version                    |   2 bytes  - required
-----------------------------------------------
|operation (request) or status-code (response)|   2 bytes  - required
-----------------------------------------------------------
|                header-tag                   |  1 byte   |
-----------------------------------------------------------|- optional
|             header-sequence                 |  k bytes  |
-----------------------------------------------------------
|              parameter-tag                  |  1 byte   |
-----------------------------------------------------------|- optional
|            parameter-sequence               |  m bytes  |
-----------------------------------------------------------
|              attribute-tag                  |  1 byte   |
-----------------------------------------------------------|- 0 or more
|            attribute-sequence               |  n bytes  |
-----------------------------------------------------------
|                 data-tag                    |  1 byte    - required
-----------------------------------------------
|                   data                      |  q bytes   - optional
-----------------------------------------------
```

The definition of all fields above is the same as defined in section 3.1 “Picture of the Encoding” except

- ‘data’ which either has the form defined in the preceding section (9.8 “Revised Picture for the Operation Layer”) or the form described in section 3.1 “Picture of the Encoding”.
- ‘header-tag’ which is new and defined like parameter-tag.
- ‘header-sequence’ which is new and defined like parameter-sequence.

9.8 Revised Syntax for the Operation Layer

The following is the revised syntax for the operation layer.

```
ipp-message = ipp-request / ipp-response
ipp-request = version operation [ header-tag  header-sequence ]
      [ parameter-tag parameter-sequence ] *(attribute-tag attribute-sequence) data
ipp-response = version status-code [ header-tag header-sequence ]
      [ parameter-tag parameter-sequence ]  *(attribute-tag attribute-sequence) data

version = major-version minor-version
major-version = SIGNED-BYTE  ; initially %d1
minor-version = SIGNED-BYTE  ; initially %d0

operation = SIGNED-SHORT  ; mapping from model defined below
```

```
status-code = SIGNED-SHORT     ; mapping from model defined below

header-sequence = *compound-header
parameter-sequence =   *compound-parameter
attribute-sequence = *compound-attribute

compound-header = header *(additional-values)
compound-parameter = parameter *(additional-values)
compound-attribute = attribute *(additional -values)

header = value-tag name-length name value-length value
parameter = value-tag name-length name value-length value
attribute = value-tag name-length name value-length value
additional-values = 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 = (data-tag  full-data ) / (chunked-data-tag chunked-data )
full-data = OCTET-STRING
chunked-data = *data-chunk END-OF-DATA
data-chunk = data-segment-length data-segment
data-segment-length = SIGNED-SHORT   ; number of octets of the data in binary
data-segment = OCTET-STRING

zero-name-length = %x00.00   ; name-length of 0
parameter-tag =  %x01            ; tag of 1
attribute-tag =  %x02            ; tag of 2
data-tag = %x03                  ; tag of 3
chunked-data-tag = %x04          ; tag of 4
header-tag = %x05                ; tag of 5
value-tag = %x10..%xFF
end-of-data = %x00.00

SIGNED-BYTE = %x00..%xFF
SIGNED-SHORT = %x00..%xFF %x00..%xFF
DIGIT = "0".."9"
LALPHA = "a".."z"
BYTE = %x00..%xFF
OCTET-STRING = *BYTE
```

10. Appendix B~~C~~: Mapping of Each Operation in the Encoding

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

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

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

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

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

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

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

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

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