INTERNET-DRAFT

HTTP 1.1 Transport Mapping for the Internet Printing Protocol

Status of this Memo

Abstract

This document describes a method to encapsulate the abstract Internet Printing Protocol (IPP) Model using the Hypertext Transfer Protocol (HTTP) Version 1.1, as described in RFC 2068 [see bib]. Abstract IPP operations and responses are mapped to specific HTTP methods and responses. Where appropriate, HTTP response codes and message headers are used to convey abstract IPP model semantics. Within the context of IPP, HTTP is primarily used as a pure transport, carrying entire IPP protocol data units (PDUs) as "payload" within HTTP messages.

1. IPP Model Overview

The IPP Model document[see bib] describes protocol operations and responses that are designed to fulfill the requirements set forth in the IPP Requirements Specification [see bib]. IPP operations are designed to operate on IPP "objects". There are two such objects defined by the IPP model: PRINTER objects and JOB objects. IPP objects are referenced by their Uniform Resource Identifier, or URI [see bib]. For each type of IPP object (PRINTER or JOB), one or more IPP operations defined. Also, for each type of IPP object, a set of attributes exist that detail object-specific characteristics or properties of that object. Individual IPP operations can be supplemented by object attribute specifications that are used to refine a particular operations' effect on a object.

2. HTTP 1.1 Mapping Strategy

IPP operations and responses can be mapped directly to HTTP methods and responses. The IPP Model Document (see bib) defines the following protocol operations:

Print-Job - Print the enclosed job, with attributes

Print-URI - Print the document specified by a URI

Create-Job - Create an instance of a JOB object

Send-Document - Append enclosed print data to an existing JOB object

Send-URI - Retrieve a document referenced by a URI and print it.

Cancel - Destroy a specific JOB object

Validate - Validate attributes for a specific PRINTER object

Get-Jobs - Return job queue information for PRINTER object

Get-Attributes - Return attribute information for a specified object

All IPP operations are encapsulated within the HTTP POST method. Each POST request and each POST response message contain message bodies that are tagged with a "Content-Type" of "application/ipp". The format of the application/ipp content-type is defined in section 3 of this document. Briefly, the application/ipp entity contains a complete IPP protocol data unit (PDU), containing all IPP-specific encodings that make up a request or response. In this sense, HTTP is being utilized as a pure transport, merely conveying IPP messages between IPP clients and IPP servers.

Associated with a POST method for IPP messages is an HTTP "Content-Length:" header that specifies the total length (in bytes) of the PDU contained within the application/ipp entity.

The absence of a Content-Length header in an IPP message indicates that the entire message is encoded using HTTP 1.1 "chunking", as described in the HTTP 1.1 specification.

2.1 IPP Status/Error Code Handling

ISSUE: In the current model document, there is a mapping of HTTP status codes to IPP status keywords. I would like to have this document define the HTTP mapping for status keywords. Is this ok?

The following information is related to the processing of response status codes received as a result of an IPP request operation. There are five different classes of status codes that can be returned by an IPP server while processing a request. Much of the description of the following classes is taken from the HTTP 1.1 specification (RFC 2068):

Informational
Success
Redirection
Client Error
Server Error

IPP responses include a 6-digit status code. The severity of the status code (information, warning, or error) is indicated by the first digit in the status code:

1xxxxx = informational, request ok, with information

2xxxxx = success, request successfully processed.

3xxxxx = redirection, further action must be taken to complete the request

4xxxxx = client error, the request contains bad syntax or cannot be fulfilled

5xxxxx = server error, the server failed to fulfill an apparently valid request

The first 3 digits of the status code are the same as that in the HTTP 1.1 header. The second group of 3-digits specifies a more detailed IPP-specific status that extends the overall HTTP 1.1 status code with more detail. The HTTP status codes, and their associated IPP status keywords are:

HTTP 1.1 Status	IPP Keyword
100 Continue	none
101 Switching Protocols	none
200 OK	successful-OK
201 Created	successful-OK
202 Accepted	successful-OK
203 Non-Authoritive Information	successful-OK
204 No Content	successful-OK
205 Reset Content	none
206 Partial Content (GET)	none
300 Multiple Choices	none
301 Moved Permanently	none
302 Moved Temporarily	none
303 See Other	none
304 Not Modified (GET)	none
305 Use Proxy	none
400 Bad Request	client-error-bad-request
401 Unauthorized	client-error-unauthorized
402 Payment Required	client-error-payment-required
403 Forbidden	client-error-forbidden
404 Not Found	client-error-not-found
405 Method Not Allowed	client-error-method-not-allowed
406 Not Acceptable	client-error-bad-request
407 Proxy Authentication Required	client-error-unauthorized
408 Request Timeout	client-error-timeout
409 Conflict (most likely PUT)	client-error-bad-request

410 Gone	client-error-gone
411 Length Required	client-error-bad-request
412 Precondition Failed	client-error-bad-request
413 Request Entity Too Large	client-error-request-entity-too-large
414 Request-URI Too Long	client-error-request-URI-too-long
415 Unsupported Media Type	client-error-unsupported-media-type
none	client-error-attribute-value-not-supported
500 Internal Server Error	server-error-internal-server-error
501 Not Implemented	server-error-not-implemented
502 Bad Gateway	server-error-internal-server-error
503 Service Unavailable	server-error-service-unavailable
504 Gateway Timeout	server-error-timeout
505 HTTP Version Not Supported	server-error-HTTP-version-not-supported
none	server-error-IPP-version-not-supported
none	server-error-printer-error
none	server-error-write-fault

RFC 2068, Section 10 should be referenced for more HTTP-related information regarding descriptions of HTTP 1.1 status codes.

3. Encoding of the application/ipp media type entity

IPP clients and servers exchange protocol messages over an HTTP message stream using the "application/ipp" media types. This document refers to the "payload" carried within an application/ipp entity as a "protocol data unit" or PDU.

The following is an example HTTP Content-Type header for IPP:

Content-Type: application/ipp; XID=FF00FF00

The parameters for the application/ipp MIME-type are 'charset' and 'XID'. The 'XID' parameter is a transaction-id that is uniquely generated by clients for request messages, and echoed by IPP servers in a corresponding response. This document specifies that all IPP entities MUST be encoded in UTF8. Consequently specifying the 'charset' parameter is optional in this version of the protocol. Likewise, if the XID parameter is not included as a Content-Type parameter, requests and responses are issued in 'lock-step' and no pipelining of IPP requests be attempted.

This section defines the encoding for an IPP protocol data unit. The application/ipp entity body contains four basic parts: IPP header, IPP operation, operation-attributes, and operation-data. The operation-data is essentially opaque to the "wire" IPP-over-HTTP protocol. Using the "8-bit clean" HTTP protocol as a transport implies that no special encoding of the actual operation-data is needed.

The syntax rules follow the Augmented Backus-Naur Form (ABNF), as specified by <insert draft name here>. Per the ABNF specification, all multi-byte tokens within this protocol specification are transmitted in big-endian format (most significant byte first).

The following protocol primitives are defined:

```
DIGIT = "0".."9"

CRLF = %d13.10

ALPHA = "A..Z"

LALPHA = "a..z"

DELIM = %x20

BYTE = %d0..%d255

OCTET-STRING = *BYTE
```

The following ABNF specification describes the encoding of an IPP message (PDU):

```
ipp-message =  ipp-version CRLF (ipp-request / ipp-response)

ipp-version = "IPP" DELIM ipp-major-version ipp-minor-
version

ipp-major-version = 2DIGIT

ipp-minor-version = 2DIGIT

ipp-request = operation CRLF *ipp-attribute CRLF *operation-
data

operation = ( "Print-Job" /          ; See section 3.1
              "Print-URI" /          ; See section 3.2
              "Create-Job" /         ; See section 3.3
              "Send-Document" /      ; See section 3.4
              "Send-URI" /           ; See section 3.5
              "Validate" /           ; See section 3.6
              "Get-Attributes" /     ; See section 3.7
              "Get-Jobs" /           ; See section 3.8
              "Cancel" /             ; See section 3.9
              "Get-Operations" )


ipp-attribute = attr-name DELIM attr-value-length DELIM
attr-value *(DELIM attr-value-length DELIM attr-value)

attr-name = 1*ALPHA *{ ALPHA / DIGIT / "_" / "-" }

attr-value-length = 1*4DIGIT
```

```
attr-value = OCTET-STRING

operation-data = OCTET-STRING



ipp-response = operation-response DELIM ipp-status CRLF
*ipp-attribute CRLF *response-data

operation-response = ( "Print-Job-Resp" /
                       "Print-URI-Resp" /
                       "Create-Job-Resp" /
                       "Send-Document-Resp " /
                       "Send-URI-Resp" /
                       "Validate-Resp" /
                       "Get-Attributes-Resp" /
                       "Get-Jobs-Resp" /
                       "Cancel-Resp" /
                       "Get-Operations-Resp"
                     )

ipp-status = 6DIGIT

response-data = OCTET-STRING
```

The first attribute in a response MAY be a "reason-phrase" and its value shall be a string which describes the status in human readable form. The value of the reason-phrase attribute MAY be localized by the IPP server according HTTP accept-header negotiation.

For a complete list of IPP attributes, as well as attribute semantics and attribute values supported by this protocol, see the IPP Model Document.

3.1 Print_Job

See IPP Model document for a description of Print-Job operation semantics.

3.1.1 Print-Job-Resp

A Print-Job Request returns a Print-Job-Resp message.

Status codes returned:

successful-OK
client-error-unauthorized
client-error-not-found
client-error-request-entity-too-large
client-error-request-URI-too-long
client-error-unsupported-media-type
client-error-attribute-value-not-supported
server-error-internal-server-error
server-error-service-unavailable

server-error-IPP-version-not-supported
server-error-HTTP-version-not-supported
server-error-

3.2 Print-URI operation

The Print-URI operation is similar in semantics to the Print-URI operation, except no operation data (print job stream) is included in the request message.

3.2.1 Print-URI-Resp

A client-generated Print-URI operation results in a Print-URI-Resp message from an IPP server.

The status codes returned in the Print-URI-Resp message include all status codes returned by the Print-Job-Resp message.

3.3 Create-Job operation

See IPP model document for a description of the Create-Job operation semantics.

3.3.1 Create-Job-Resp

A Create-Job Request returns a Create-Job-Resp message. The syntax and semantics of the Create-Job-Resp are identical to that of the Print-Job-Resp, with the exception that no response data is returned.

3.4 Send-Document Request

See IPP Model document for a description of Send-Document operation semantics.

3.4.1 Send-Document-Resp

A Send-Document Request returns a Send-Document-Resp message. The following HTTP status codes MAY be returned in a Send-Document-Resp message:

3.5 Send-URI Request

See IPP Model document for a description of Send-URI operation semantics.

3.5.1 Send-URI-Resp

A Send-URI request generates a Send-URI-Resp message from the server. The syntax and status code semantics for the

Send-URI-Resp message are identical to the Print-URI-Resp message.

3.6 Validate Request

See IPP Model document for a description of Validate operation semantics.

3.6.1 Validate-Resp

A Validate Request generates a Validate-Resp message from an IPP server.

The HTTP status codes that MAY be returned in a Validate-Resp message are:

3.7 Cancel Request

See IPP Model document for a description of Cancel operation semantics.

3.7.1 Cancel-Resp

A Cancel Request generates a Cancel-Resp message from an IPP server. The Cancel-Resp message can contain an attribute set wherein the following attributes MAY be specified:

job-state
job-state-reasons
job-state-message
job-k-octets-completed
job-impressions-completed
job-media-sheets-completed
time-since-submission
time-since-processing
job-originating-user
job-originating-host

The following HTTP status codes MAY be returned in a Cancel-Resp message

3.8 Get-Attributes Request

See IPP Model document for a description of Get-Attribute operation semantics.

3.8.1 Get-Attributes-Resp

A Get-Attributes Request generates a Get-Attributes-Resp message from an IPP server.

SAY THAT ALL REQUESTED ATTRIBUTES ARE RETURNED, AND THAT UNSUPPORTED ATTRIBUTES ARE MARKED AS "UNSUPPORTED".

3.9 Get-Jobs Request

See IPP Model document for a description of Get-Jobs operation semantics.

3.9.1 Get-Jobs-Resp

A Get-Jobs Request generates a Get-Jobs-Resp message from the IPP server.

Status codes returned:

4.0 HTTP 1.1 General Headers

Many of the headers specified by RFC 2068 do not have to be supported by general purpose HTTP clients and servers. The following text clarifies what IPP clients and servers should consider and the HTTP 1.1 conformance issues for each.

4.1 Cache-Control

IPP servers and clients will have to operate within a caching proxy environment. In order to ensure a pure client and server environment between IPP clients and servers, caching of IPP requests and responses must be prohibited. The "Cache-Control:" general header MUST be included in all IPP requests and responses. The value for the "Cache-Control:" header directive would be "no-cache". Also, to make sure there are no HTTP 1.0 caching proxies between HTTP 1.1 clients and servers, IPP clients and servers MUST also include the "Pragma:" general header, also specifying "no-cache" as the value.

It is conceivable that there would be some value in caching of attribute requests to IPP printer objects, since in a large environment, these requests might be very frequent. The IPP working group may want to consider the value in caching certain IPP object attribute requests. If caching of certain IPP response data is allowed, then we should also consider the use of the "no-transform" value for the "Cache-Control" directive.

4.2 Connection

The "Connection:" general header SHOULD be used by IPP servers or IPP clients to instruct either a remote client or server that the HTTP connection be closed. For an IPP server that accepts CREATE-JOB/SEND-DOCUMENT sequences for job submission, the last SEND-DOCUMENT (POST) operation required to deliver the job data could include the "Connection:"

header with the value "close" to instruct the server that the connection will be closed after reception of this request.

4.3 Content-Coding

The "Content-Coding:" entity header field specifies how the entity body of a particular message is to be decoded. For HTTP, this is typically a compression encoding so the field would be "gzip" or "compressed". If the IPP working group wants to define a base set of content-codings, then the values for these codings would be specified in IPP messages via the "Content-Coding:" header.

Content-Language:
Like the "Content-Coding:" header, this header also specifies information related to the enclosed entity. The "Content-Coding:" header describes the natural language in which the entity body has been encoded. IPP servers should only return entities in languages that have been "agreed" upon by a particular client in a previously received "Accept-Language:" request header.

4.4 Content-Length

The "Content-Length:" header specifies the size of a message body. IPP clients and servers will use the same algorithm as general-purpose HTTP 1.1 servers for determining the length of IPP messages. From RFC 2068, the description of the Content-Length header:

Applications SHOULD use this field to indicate the size of the message-body to be transferred, regardless of the media type of the entity. It must be possible for the recipient to reliably determine the end of HTTP/1.1 requests containing an entity-body, e.g., because the request has a valid Content-Length field, or is chunked.

4.5 Content-Type

The Content-Type header would be used by IPP clients and servers to specify IPP-specific entities. The Content-Type value MUST be “application/ipp”.

4.6 Date

The “Date:” header field is currently specified in the HTTP 1.1 document as a MUST header by all compliant implementations. The date format used as the value of this header must be in RFC 1123 format. There is a recent internet draft that has been published that attempts to describe how some embedded, lightweight HTTP server implementations can still be “compliant” even if they don’t contain any realtime clock or time capabilities.

4.7 Pragma

The “Pragma:” directive would only be used by IPP implementations for backwards compatibility with HTTP 1.0 caching proxies. The Pragma header would specify the value “no-cache”, which is understood by HTTP 1.0 proxies to have the same semantics as the HTTP 1.1 “Cache-Control” directive with the value “no-cache”.

4.8 Transfer-Encoding:

For HTTP 1.1, the only “Transfer-Encoding” specified is the “chunked” encoding. Since an HTTP connection is “8-bit clean”, the traditional rationale for transfer-encodings (like used in MIME) are unneeded. But when IPP implementations are attempting to send messages for which the total length of the message cannot be determined, then the message should be transferred as "chunked" or via a multipart message with message boundaries. The current HTTP specification requires that all HTTP 1.1 applications MUST be able to receive and decode the chunked transfer encoding.

5.0 HTTP 1.1 Request Headers

5.1 Accept

The Accept: header is used to specify certain media types that a client is willing accept as a result of a request to a server. IPP clients SHOULD always specify (at a minimum) application/ipp, text/html, and text/plain.

5.2 Accept-Charset

This header indicates to servers what character sets a client is willing to accept in a response. According to the HTTP 1.1 specification, all clients should be able to support ISO-8859-1.

5.3 Accept-Encoding

Similar to "Accept:", the Accept-Encoding header is sent from client to server to inform the server what types of encoding of responses that the client can handle.

5.4 Accept-Language

IPP clients SHOULD send Accept-Language headers in IPP requests to notify IPP servers what type of localization is acceptable to the client.

5.5 Authorization

IPP servers may protect certain types of IPP objects via HTTP basic authentication. If an IPP client has knowledge that a requested resource requires basic authentication, then an appropriate "Authorization:" request header should be included in all IPP requests to the IPP object (URI) in question. The client can also dynamically learn of the authentication requirements for a particular object if the client attempts to access the object without an authentication header. IPP servers that receive un-authenticated requests for IPP objects that require basic authentication would return a status code of 401, which indicates to clients that authentication is required for accessing the requested object.

It is assumed that, for the lifetime of a particular IPP object (URI), that the user’s credentials (once successfully validated) will be valid. Therefore, on the first successful authenticated response to a request, IPP clients can cache the user’s credentials and reuse these credentials on subsequent requests to the server for this object. Each subsequent request for the IPP object (URI) would include an "Authorization:" header specifying the cached credentials.

5.6 From

The "From:" header contains the internet e-mail address for the human individual that is responsible for the request being generated. The IPP working group has talked about using the “From:” header as a means for some type of authentication or access protection. The current HTTP 1.1 specification states that the "From:" header “SHOULD NOT be used as an insecure method of access protection”. The specification goes on to say that “the interpretation of this field is that the request is being performed on behalf of the user specified by the "From:" header, who accepts responsibility for the operation being performed.”. The following paragraph from RFC 2068 is especially relevant:

Note: The client SHOULD not send the From header field without the user's approval, as it may conflict with the user's privacy interests or their site's security policy. It is strongly recommended that the user be able to disable, enable, and modify the value of this field at any time prior to a request.

5.7 Host

The "Host:" field typically comes on a separate line after the HTTP method specification. This field MUST be set by HTTP 1.1 clients with the network location of the specified URI in the method. All internet-based HTTP 1.1 servers MUST respond with a 400 status code to any HTTP 1.1 request message which lacks a "Host:" header. This header is used by newer WEB server sites for so-called "virtual host" access. IPP could utilize this field in some very interesting ways with regards to multiple logical printers serviced by a single IPP/HTTP server.

5.8 Proxy-Authorization

When there is an HTTP 1.1 caching proxy operating in between an IPP client and server, it is possible that certain resources identified by a site administrator might require basic authentication. If an IPP client receives a 407 response to a valid IPP request, the client should format an authorization request back to the requested resource (URI) using the "Proxy-Authorization:" request header. Section 11 of RFC 2068 discusses HTTP authorization in detail.

6.0 HTTP 1.1 Response Headers

6.1 Allow

The "Allow:" entity header field MUST be returned by IPP servers to notify IPP clients which HTTP methods are allowed to be executed on a particular URI (or IPP object). In the future, we may want to define conformance levels with respect to IPP, wherein some IPP servers implement all possible methods on IPP objects, and other lighter weight IPP servers are restricted in the domain of methods supported on IPP objects. The "Allow:" header permits interoperability between clients and servers of different capabilities. The client can adapt its behavior to the capabilities it learns from a particular server.

6.2 Location

The "Location:" header MAY be used by IPP servers to dynamically redirect IPP clients to other URIs that can be contacted for completing the client’s request. The Location header could be used as a replacement for the multiple-URL facility discussed in the early IPP-over-HTTP internet draft. IPP implementations would follow the direction set forth by the current HTTP 1.1 specification:

"for 201 (“Created”) responses, the “Location” is that of the new resource created by the request. For 3xx responses, the location SHOULD indicate the server’s preferred URL for automatic redirection to the resource."

The term "resource" used in the above paragraph would normally be a URI referencing an IPP job object.

6.3 Proxy-Authenticate

It is possible that, in the presence of caching HTTP 1.1 proxies, that IPP client implementations may have to deal with "Proxy-Authenticate" responses. The "Proxy-Authenticate" response header could be returned as part of a 407 (Proxy Authentication Required) response. (see also Proxy-Authorization request header). IPP clients SHOULD support proxy-authorization.

6.4 Public

The "Public:" response header SHOULD be used by IPP servers to inform IPP clients what types of HTTP methods are supported by the server. The "Public" response header would typically be used by very lightweight HTTP/IPP server implementations that implement a minimal IPP capability.

6.5 Retry-After

The “Retry-After” response header would be used in tandem with the 503 (Service Unavailable) response code to indicate how long the resource (or service) is to remain unavailable. This could be used by IPP servers to indicate how long a printing service might be unavailable to IPP clients.

6.6 WWW-Authenticate

The WWW-Authenticate response header is used to initiate basic HTTP authentication. If an IPP client receives a 401 (Unauthorized) response to an IPP request, then the response MAY contain a "WWW-Authenticate" header with an appropriate challenge. The next request for this resource formulated by the IPP client SHOULD contain an "Authorization" header specifying appropriate credentials.

7.0 Security Considerations

When utilizing HTTP 1.1 as a transport for IPP, all of the security considerations specified in RFC 2068 apply. In addition, the IPP adds some additional application-specific security considerations, including denial-of-service attacks, mutual authentication, and privacy. The IPP Model document addresses IPP-specific security considerations, while RFC 2068 addresses HTTP-related security considerations.

ISSUE: the security subgroup is free to add whatever is necessary to fill out the "security considerations" section of this document. However, the IPP model document should include the bulk of security discussions that are IPP-specific.

Appendix A - Transport Requirements

Some transports, such as raw TCP/IP, don’t have a way to specify length or carry along attributes that pertain to the transport. An example of such an attribute is the compression encoding for the operation. In addition, the target URI needs to be specified as a transport attribute. This section specifies the encoding for such cases.
The encoding is defined as follows:

transport-encoding = transport-version transport-attributes transport-operation

transport-version = transport-major-version transport-minor-version

transport-major-version = one-byte-integer ; major version in binary, initially 1 transport-minor-version = one-byte-integer ; minor version in binary, initially 0

transport-attributes = attribute-length *attribute

transport-operation = (type-zero operation-encoding) | (type-one 1*transport-chunk)

type-zero = byte-value-zero ;; a byte with binary 0

type-one = byte-value-one ;; a byte with binary 1

transport-chunk = transport-data-length transport-data

transport-data-length = four-byte-integer ; number of octets of the data in binary

transport-data = octet-string

The transport-version shall be the version for this syntax and is separate from the operation-version. A zero length transport-chunk shall denote the end of the transport-operation. The transport-data contains fragments of the operation. When all the transport-chunks are concatenated together, they shall form a complete operation.
The first attribute in the transport-header for a request shall be the attribute “URI” which is the target object of the operation.

ISSUE: is the URI sufficiently important that it should have its own field that doesn’t require looking for the URI attribute.
ISSUE: should the status and reason-phrase be in the transport-header instead of the operation-header for the responses?
ISSUE: is the type-zero and type-one encoding worth the complexity to save specifying the same length twice for data-less operations – once at the transport level and once at the operation level. The type-one encoding is the most compatible with HTTP.
The transport-operation may use type-zero encoding for operation-encodings with no operation-data.

References

[RFC 2068]
[IPP Model Document]