A proposed mapping of IPP onto XML

Author: Paul Moore, Microsoft Corporation

Abstract:

This document details a method of expressing the current IPP model and semantics definition [MODEL] in XML.

It is assumed that the reader is familiar with the current set of IPP proposals.

1 Basic Encoding Rules

IPP requests are encoded as HTTP messages sent by POST. The responses to those requests are delivered in the HTTP reply to the POST.

HTTP 1.1 chunked encoding may be used in the case where the message length is not known in advance.

The HTTP POST carrying the IPP protocol is normally multi-part MIME format. If the IPP request contains no raw data (PDL for example) then it may be sent using MIME type text/xml.

The response rules are the same; if raw data is present then multi-part MIME must be used, otherwise it is optional.

There will always be at least one text/XML part. There may be additional parts, in particular there may be an opaque section containing the PDL.

```
Content-type:multipart/mixed;
Boundary=”A string That does not occur elsewhere”;
--A string That does not occur elsewhere
Content-Type: text/xml
[IPP request]
--A string That does not occur elsewhere
Content-Type: Application/octet-string
[PDL]
--A string That does not occur elsewhere—
```

A request or response is contained within one XML block. (This allows multiple requests to be encoded in the future)

```
<?xml version=”1.0” encoding=”UTF-8”>
<request> or <response>
…….
</request> or </response>
```

Nothing may appear outside the request/response block within a given multi-part sub-block.

The character set is always UTF-8. The mandatory attribute attribute-charset is thus not included in any request or response.

Attributes are expressed as XML elements (as opposed to XML attributes). XML attributes are used to qualify or modify the meaning of a given attribute (language, type or character set for example). Note the potential for confusion here – XML uses the term ‘attribute’ so does [MODEL]. This document will refer to IPP attributes as ‘attributes’ and XML attributes as ‘XML attributes’.

IPP Attribute names as XML element names and XML attributes names as qualifiers are case sensitive (following XML standards) and must therefore be expressed exactly as they are defined in [MODEL]. All other strings have case sensitivity as defined in [MODEL].

Leading and trailing spaces are significant in all values. This may mean that they are invalid in some contexts (numeric strings for example).

No ordering of attributes is implied or required other than that explicitly stated.

2 Request encoding

A request always specifies the operation, the request-id, language and the version of that operation.

A response always specifies the operation and request-id to which it is a response, the language and the version of that response.

These attributes must appear before any other attributes. They may appear in any order.

```
<request>
      <operation>print-job</operation>
      <version>1.0</version>
      <request-id>123</request-id>
      <attributes-natural-language>en</attributes-natural-
      language>
      [remainder of request]
</request>
```

2.1 Operation attributes

Attributes that apply to the request directly (these were once called parameters) appear at the same structure level as the operation and version attributes.

```
<request>
      <operation>get-jobs</operation>
      <version>1.0</version>
      <request-id>123</request-id>
      <attributes-natural-language>en</attributes-natural-
      language>
      <my-jobs>1</my-jobs>
</request>
```

3 Attribute mapping

An attribute is encoded as an XML element. The value is the element value. All values are expressed as character strings which may have to be translated by the receiver into the appropriate type (integer for example).

The element name is the name of the attribute as specified in [MODEL]

Empty values are represented by an empty element. This includes the out of band values ‘unknown’ and ‘no-value’.

The out of ban value ‘unsupported’ does not have any explicit encoding. The only place it occurs in [MODEL] is in the psuedo-1setof attribute ‘Unsupported-Attributes’ – the ‘unsupported’ nature of the value is therefore implied by the context.

3.1 Qualification

An attribute value may be qualified by the use of XML attributes for the element. The following qualifiers are defined: -

Type: This defines the type of the element (Integer, Boolean, etc.) as defined in the model document. This is optional and may be used to disambiguate. The type values themselves are expressed as Keywords as defined in [MODEL]. Eg type=integer or type=Keyword

Lang: This specifies the human readable language of the attribute. This is optional for all name and text types. Including it means that the attribute becomes and xxxWithLanguage as opposed to a plain xxx. The representation of the language name is that defined in [MODEL] (eg RFC1766). Example ‘lang=fr’

Qualifiers themselves are a type4 keyword.

Examples: -

Foo1 is a text attribute, value “bar”

```
<Foo1>bar</foo1>
```

Same, but language is overridden to ‘de’

```
<foo1 lang=”de” >bar </foo1>
```

Same, but no value

```
<foo1> (or should it be <foo1></foo1>)
```

Foo2 is an integer, value 1

```
<foo2>1</foo2>
```

Same, but type given explicitly

```
<foo2 type=”Integer”>1</foo2>
```

The following is invalid because integer types do not have language associated with them.

```
<foo2 type=”Integer” lang=”fr”>1</foo2>
```

3.2 Simple types mapping

The following types are represented as strings: -

- Name (and namewithlang)
- Text (and textwithlang)
- Keyword
- Uri and urischeme
- Charset
- Natural language
- MIME media type

The following types are represented as integer numeric strings

- Integer
- Enum
- Boolean (1= true, 0=false)

An integer numeric string is a sequence of ASCII characters representing the digits 0 thru 9, ‘+’ and ‘-‘. Leading and trailing spaces may not be present. Leading zeroes are permitted and are ignored. The ‘+’ or ‘-‘ character may only appear in the first character position, if omitted then the values is treated as positive. No signed attributes are currently defined in the model and hence a leading ‘-‘ would be invalid.

Datetime is represented as a the ‘hinted’ string as defined in RFC1903 i.e “1992-5-26,13:30:15.0,-4:0”

Octet-string is represented as base64.

4 Object Attributes

Attributes that are associated with an object are nested within an XML block. An object in this sense is a construct which may be referenced as a whole and where multiple instances of it may occur in a message. The only example of this in the current model is job. Future extensions to IPP may require other objects such as document, printer, server, font, user, etc.

The attributes contained within the block are those defined in [MODEL] as being attributes of that object (e.g. job attributes).

A job is represented as follows.

```
<request>
      <operation>print-job</operation>
      <version>1.0</version>
      <attributes-natural-language>en</attributes-natural-
      language>
      <job>
            <job-name>My print Job</job>
            <job-impressions>2</copies>
      </job>
</request>
```

If multiple instances of a given object are included in a message then they are simple concatenated into a list. Example: -

```
<response>
      <operation>get-jobs</operation>
      <version>1.0</version>
      <request-id>1</request-id>
      <attributes-natural-language>en</attributes-natural-
      language>
      <job>
            <job-name>job1</name>
            <job-id>1234</id>
      </job>
      <job>
            <job-name>job 2</name>
            <job-id>2222</id>
      </job>
</response>
```

5 Compound and meta-type mapping

Structured types are represented as XML blocks. The outer element is has the name of the attribute as its name. The contained elements have the name of the sub-components of the type.

5.1 Resolution

```
<Printer-resolution>
      <x>600</x>
      <y>600</y>
</printer-resolution>
```

5.2 Range of Integer

```
<job-impressions-supported>
      <from>0</from>
```

```
      <to>20</to>
<job-impressions-supported>
```

5.3 1Setof

A set of values is represented as an XML block. The outer block name is the attribute name, the member values are each contained within an element call ‘member’. The attribute value itself is expressed identically to any other attribute. In particular the same qualifying XML attributes may be used.

```
<RequestedAttributes>
      <member>job-name</member>
      <member>job-id</jobid>
</RequestedAttributes>
```

5.4 Attribute groups

The main example of this is ‘unsupported-attributes’. These are represented as a concatenated list of the values that were in error: -

```
<unsupported-attributes>
      <job-impressions>300</job-impressions>
      ….
</unsupported-attributes>
```

An alternative, more consistent way of representing them would be to treat these as a response attribute of type 1setof of attributes. For example:-

```
<unsupported-attributes>
      <member>
            <job-impressions>300</job-impressions>
      </member>
      ….
</unsupported-attributes>
```

This is not adopted here because it would be model change.

6 Examples

6.1 Print-job

Request, including copies, duplex and name.

```
Content-type:multipart/mixed, Boundary=”A string That does not
occur elsewhere”
--A string That does not occur elsewhere
Content-Type: text/xml
<?xml version=”1.0” encoding=”UTF-8”>
<request>
      <operation>print-job</operation>
      <version>1.0</version>
      <request-id>1</request-id>
      <attributes-natural-language>en</attributes-natural-
      language>
      <job>
            <job-name>My print Job</job>
```

```
            <job-impressions>2</copies>
            <sides>two-sided-long-edge</sides>
      </job>
</request>
A string That does not occur elsewhere
Content-Type: Application/octet-string
[PDL]
--A string That does not occur elsewhere—
```

6.2 Print-job Response (OK)

```
Content-Type: text/xml
<?xml version=”1.0” encoding=”UTF-8”>
<response>
      <operation>print-job</operation>
      <version>1.0</version>
      <request-id>1</request-id>
      <attributes-natural-language>en</attributes-natural-
      language>
      <status>OK</status>
      <status-message>OK</status-message>
      <job>
            <job-id>123</job-id>
            <job-uri>http://foo/bar/123</job-uri>
            <job-state>pending</job-state>
      </job>
</response>
```

6.3 Print-job Response (Failed)

```
Content-Type: text/xml
<?xml version=”1.0” encoding=”UTF-8”>
<response>
      <operation>print-job</operation>
      <version>1.0</version>
      <request-id>1</request-id>
      <attributes-natural-language>en</attributes-natural-
      language>
      <status>Client-error-bad-request</status>
      <status-message lang=fr>Dommage, mauvais requet</status-
      message>
      <unsupported-attributes>
            <job-k-octets<123456789</job-k-octets>
            <job-impressions>20</job-impressions>
      </unsupported-attributes>
</response>
```

6.4 Print-URI request

Show not using multi-part MIME.

```
Content-Type: text/xml
<?xml version=”1.0” encoding=”UTF-8”>
<request>
      <operation>print-URI</operation>
      <version>1.0</version>
```

```
        <request-id>1</request-id>
        <attributes-natural-language>en</attributes-natural-
        language>
        <job>
              <job-name>My print Job</job>
              <job-impressions>2</copies>
              <sides>two-sided-long-edge</sides>
              <document-URI>ftp://foo.com/foo</document-URI>
        </job>
  </request>
```

6.5 Create-Job request

No job object appears because no job attributes are given. [Alternatively an empty job object could be supplied]

```
  Content-Type: text/xml
  <?xml version="1.0" encoding="UTF-8">
  <request>
        <operation>create-job</operation>
        <attributes-natural-language>en</attributes-natural-
        language>
        <version>1.0</version>
        <request-id>1</request-id>
  </request>
```

6.6 Get-Jobs request & response

Request, limit to 50 but no filtering. Return job-id, job-name and document-format.

```
  Content-Type: text/xml
  <?xml version="1.0" encoding="UTF-8">
  <request>
        <operation>get-jobs</operation>
        <version>1.0</version>
        <request-id>1</request-id>
        <attributes-natural-language>en</attributes-natural-
        language>
        <limit>50</limit>
        <Requested-attributes>
              <member>job-id</member>
              <member>job-name></member>
              <member>document-format</member>
        </requested-atrributes>
  </request>
```

6.7 Response, 2 jobs returned.

```
  Content-Type: text/xml
  <?xml version="1.0" encoding="UTF-8">
  <response>
        <operation>get-jobs</operation>
        <version>1.0</version>
        <request-id>1</request-id>
        <job>
```

```
            <job-id>147</job-id>
            <job-name>fou</job-name>
            <document-format>application/postscript</document-
      format>
      </job>
      <job>
            <job-id>148</job-id>
            <job-name lang="de-CH">isch guet</job-name>
            <document-format>text/plain</document-format>
      </job>
</request>
```

7 Use of DTD

No DTD is required to validate the protocol on the receiver's side.

Rationale: This would constrain the protocol from being further extended. Plus it does not really add anything; the receiver must validate the request, it is merely a matter of which piece of code on the receiver does the validation. In addition, a DTD based validation, being generic, would be more expensive to implement, this is a significant consideration for people embedding IPP in printers.

Note that a given implementation MAY choose to use a DTD based validation method (using a private DTD that expresses the exact protocol supported by a given receiver) but this is a private issue; the point is that there will be no standardized DTD or set of DTDs issued.

8 A full OO representation

It is worthwhile looking at how the XML representation of data could be used for a more explicit object oriented approach. The following issues need to be addressed:

- A rationalization of attribute naming
- The explicit inclusion of objects that lurk in the background of the IPP model (document, printer, etc.)
- The specification of every attribute as belonging to an object – i.e. there sre no 'floating' attributes that don't belong to an object.
- The nesting of objects

The attribute name space in [MODEL] is flat. There is an attempt to add some structure by qualifying many names with the object to which they apply, document-format, job-name, etc. However this is not consistently used, for example number-of-documents is a job attribute.

A more OO approach would be to explicitly assign an object name to each construct in the model and explicitly qualify the name using that object. We would then have Request.charset, Job.name, Document.format, etc.

The second step is then to explicitly state a containment hierarchy. For example a request may contain one or more jobs, a job may contain one or more documents, etc.

Finally one would specifically indicate that certain attributes have objects as their value, for example the document attribute of a job is a document object (or alternatively the Documents attribute is 1setof of document objects).

An abbreviated print-URI example: -

```
<request>
      <operation>print-URI</operation>
```

```
            <lang>en</lang>
            <job>
                  <copies>1</copies>
                  <name>my job</name>
                  <document>
                        <URI>ftp://foo.bar/foo</URI>
                        <format>postscript</format>
                  </document>
            <job>
      </request>
```

Or possibly: -

```
      <request>
            <operation>print-URI</operation>
            <job>
                  <copies>1</copies>
                  <name>my job</name>
                  <documents>
                        <member>
                              <document>
                                    <URI>ftp://foo.bar/foo</URI>
                                    <format>postscript</format>
                              </document>
                        </member>
                  </documents>
            <job>
      </request>
```

An interesting question would be whether or not one would require a name for each object instance (for example request-ID is required but not job or document. This would be useful in the situation where there are multiple instances of an object in a message. For example if there were >1 document in a job then how could the server indicate which document it is refering to in a message. If each obejct instance had an ID then no ambinguity would exist. For example a document might be 123.1.1 (request.ID, Job.ID, Document.ID), the URI attribute would be 123.1.1.URI.

Filename:	draft-xml4ipp-word97.doc
Directory:	C:\jkm\PWG\IPP\microsoft
Template:	C:\Program Files\Microsoft Office\Templates\Normal.dot
Title:	XML For IPP
Subject:	
Author:	paulmo
Keywords:	
Comments:	
Creation Date:	02/08/98 3:53 PM
Change Number:	6
Last Saved On:	02/11/98 9:13 AM
Last Saved By:	paulmo
Total Editing Time:	59 Minutes
Last Printed On:	02/11/98 2:29 PM

As of Last Complete Printing

Number of Pages:	11
Number of Words:	2,478 (approx.)
Number of Characters:	14,127 (approx.)