PWG DRAFT
File: ipp-collection-attr-syntax-v096.doc

Roger deBry
IBM Printing Company
T. Hastings
Xerox Corporation
R. Herriot
Sun Microsystems
August 6, 1998

Proposed Internet Printing Protocol/1.0 Extension 'collection' attribute syntax

Status of this Memo:

This document is a PWG Working Draft. It proposes an OPTIONAL extension to the IPP/1.0 Model and Semantics document [ipp-mod]. There are some issues in Sections 4.1, 5, and 7. This attribute syntax will be registered with IANA after approval by the WG and after IPP/1.0 has been published as RFCs. We may want to publish it as an RFC as well. This attribute syntax had originally been named 'dictionary'. We agreed to change its name to 'collection' in [ipp-pro], since the member attributes are not ordered, typically. [ipp-pro] has reserved the tag value code 0x34 for 'collection'. Some future extensions, both registered and private, can make use of this new attribute syntax.

Abstract

This document specifies a new attribute syntax called 'collection'. A 'collection' value is itself a set of attributes, called "member" attributes, that are grouped together as the value of an attribute. The member attributes may be SINGLE-VALUED or MULTI-VALUED (1setOf). An attribute that uses the 'collection' attribute syntax may be SINGLE-VALUED ('collection) or MULTI-VALUED ('1setOf collection') as well.

Table of Contents

1 Problem Statement

There is no good way to add attributes that contain several fields, whether the fields are mandatory or optional. Instead of each new attribute that needs more than one field (struct), requiring a new ad hoc attribute syntax, such as we have done for the 'resolution' attribute syntax for use in the "printer-resolution" attribute, it would be desirable to have a simple, general mechanism for representing multi-field values. (ISO DPA [ISO-10175] also had many ad hoc syntaxes for structure data types using ASN.1) It would also be desirable to allow fields to be omitted, when the attribute specification allows that. This mechanism would be useful for both new attributes that we might register as extensions to be used with the IPP standard, or that implementers might implement as private extensions.

2 Summary of the attribute syntax alternative

A number of other alternatives were considered. See the last section for a list and the reasons for their rejection.

The proposal is to add a new attribute syntax, called 'collection'. Any attribute of type 'collection' ha a value that is a set of attributes, called "member" attributes. Each member attribute MAY be single-valued or multi-valued (1setOf collection) as specified for the attribute that has the 'collection' attribute syntax. Since each attribute value has a length like any other attribute value, IPP objects not supporting the attribute can easily skip over the entire attribute value, i.e., skip over the entire set of attributes that make up a collection value.

3 Requirements for and properties of the suggested collection mechanism

The collection mechanism for use with IPP needs to have the following semantic properties:

1. The collection mechanism provides a way to supply and query a set of attributes as a logical unit. Then each 'field' that is present in the collection would be self-identifying by its attribute name.

2. The attributes in a collection are unordered. Therefore, an IPP object MUST be able to accept attributes in a collection in any order. In order to improve processing efficiency, one or more member attributes of the collection may be specified as being REQUIRED to be first, just as for operation attributes in an IPP request.

3. The semantics of a collection attribute specifies which attributes in a collection instance are REQUIRED for the IPP object to support and which are OPTIONAL for the IPP object to support when the IPP object supports that collection attribute.

4. The semantics of a collection attribute specifies which attributes in a collection instance are required for the requester to supply and which the requester may omit.

5. A collection attribute could be single valued, i.e., with one collection value consisting of a set of attributes, or could be multi-valued, i.e., with multiple collection values, each consisting of a set of attributes.

6. An attribute in a collection value can be single valued or multi-valued as well according to the specification of the collection attribute.

7. As with all attribute values, if an IPP object does not support a collection attribute, it must be easy for the IPP object to ignore each collection attribute value, including returning whatever is required in the Ignored Attributes group in the response.

8. The syntax of each collection value is the same as a group of attributes in a request or response, so each attribute in a collection value instance has its keyword name, its attribute syntax code, and its value.

9. An implementer MAY support additional registered or private attributes in a collection. In other words, a collection is extensible, just like an attribute group in an operation or response.

10. Since support of all possible combinations of values for all attributes in a collection value may not be supported by some implementations, there should be a way for the IPP object to indicate which combinations of values are supported. For example, 300x300, 600x300, and 600x600, but not 300x600 dpi.

11. Finally, an attribute in a collection value can be itself a collection, so that nesting could be allowed, if the specification of a collection attribute allowed a collection attribute to be contained in its collection.

4 Examples of collection usage

This section describes four collection Job Template examples: "printer-resolution", "job-notify", "job-start-page-contents", and "postal-mail-disposition" attributes. The "printer-resolution" and "job-notify" attributes only contain single-valued member attributes, while the "job-start-page-contents" and "postal-mail-disposition" attributes contain multi-valued member attributes.

4.1 Example a: "printer-resolution" Job Template attribute

For example, the new "printer-resolution" attribute was defined using a very ad hoc 'resolution' attribute syntax. Had we had the collection attribute syntax, we might have chosen to use it here, though we wouldn't have had to either. If we did use the 'collection' attribute syntax for the "resolution", the attribute value would contain the following attributes: "resolution", "cross-feed-resolution", and "resolution-units". We could have also specified that the "cross-feed-resolution" attribute is OPTIONAL and when omitted, the cross-feed resolution is the same as the "resolution" attribute, since most resolutions

are the same in both directions. We could have also specified that the "resolution-units" attribute is OPTIONAL and when omitted, the resolution units are dots per inch.

For the resolution, the "resolution" member attribute may be supplied by the client by itself when the value is the same for feed and cross-feed and the units are dots per inch. This would allow simple clients to provide most of the resolution capability in a simple way.

The specification for the "printer-resolution" collection attribute is that its collection value is made up of the following attributes:

Attribute name	syntax	in request
"resolution"	integer	required
"cross-feed-resolution"	integer	optional
"resolution-units"	enum	optional

For a simplified collection attribute notation, lets use:

"*collection attribute*" = { *set of attributes and values* }

where a set of {} is used to group a single collection value.

For example, a client supplying a resolution of 600 x 300 would be indicated in examples using the following notation:

"printer-resolution" = { "resolution" = '600', "cross-feed-resolution" = '300' }

4.1.1 "printer-resolution-default" example

The Printer object could represent the "printer-resolution-default" default values as a single collection value. For example, a system administrator (or the printer vendor) could specify the default as:

"printer-resolution-default" = { "resolution" = '300' }

4.1.2 "printer-resolution-supported" example and validation of collections

The Printer object could indicate the combinations of resolutions that are supported by three sets of collection values which represent 300x300, 600x300, and 600x600 dpi, respectively (300x600, say, is not supported). Such a configured situation could be represented in examples as:

```
"printer-resolution-supported" = {
        { "resolution" = '300' },
        { "resolution" = '600', "cross-feed-resolution" = '300' },
        { "resolution" = '600' } }
```

4.2 Example b: "job-notify" Operation attribute

NOTE: The current proposal for notification does not use the collection mechanism [ipp-not]. This example just shows how we could use the collection attribute syntax, if it is necessary to be able to group events and methods, rather than saying that the mail method ignores most of the events, so that other methods can be specified in the same job subscription. Because the 'collection' attribute syntax is itself multi-valued, the member attributes do not need to be, thus simplifying the syntax However, the same recipient can be in more than one collection value, and the same event group can be in more than one collection value.

In order to allow a client to supply different event groups for different recipients/methods, the requester must be able to supply one or more notification collection values, where each collection value consists of one "notify-event" attribute and one "notify-recipient" attribute. Additional registered or private attributes may be included in the future. There might be a similar multi-valued "printer-notify" Printer object collection attribute that is supplied by a new Subscribe operation, but is independent of jobs. Both the "job-notify" and the "printer-notify" collection attributes are MULTI-VALUED but contain attributes that themselves are only SINGLE-VALUED.

The "job-notify" Operation collection attribute would have collection values with the following syntax:

Attribute name	syntax	in request
"notify-event-group"	type2 keyword	OPTIONAL
"notify-recipient"	uri	REQUIRED

A Print-Job request could supply the collection attribute values in order to send immediate 'job-error' events to Smith (himself) and e-mail 'job-completion' to Jones and White.

```
"job-notify" =   {     "notify-event-group" = 'job-errors'
                       "notify-recipient" =
                       "ipp-tcpip-socket:13.240.120.138/port=6000' },
                 {     "notify-event-group" = 'job-completion'
                       "notify-recipient" = 'mailto:Jones' }
                 {     "notify-event-group" = 'job-completion'
                       "notify-recipient" = 'mailto:White' }
```

4.3 Example c: Start page fields supplied by the end-user

As a third example of a collection, an attribute could represent the fields that the submitter wishes to be printed on the job-start page. The name of the attribute might be: "job-start-page-contents". The collection value might include: "job-name", "user-name", "job-comment", "account-name", "job-disposition", "job-delivery", etc. where the values of the attributes in the collection are printed after each attribute name on the job-start-page.

Attribute name	syntax	in request
"job-name"	name	required
"user-name"	name	required
"job-comment"	text	optional
"account-name"	name	optional
"job-disposition"	keyword	optional
"job-delivery"	1setOf keyword	optional

4.4 Example d: Postal mailing address

As a final example of a collection, an attribute could represent a postal mailing address for the output. The name of the attribute might be "postal-mail-disposition" and it would be multi-valued, i.e., 1setOf collection. The collection attribute might have the following specification and support requirements if the "postal-mail-disposition" collection attribute is supported at all:

Attribute name	syntax	in request	IPP object support
"addressee-name"	text	required	REQUIRED
"company-name"	text	optional	OPTIONAL
"internal-mail-stop"	text	optional	OPTIONAL
"apartment-number	text	optional	REQUIRED
"street-address"	text	required	REQUIRED
"city-or-town	text	required	REQUIRED
"state"	text	required	REQUIRED
"postal-zone	text	required	REQUIRED
"country"	text	optional	OPTIONAL
"phone-numbers	1setOf text	optional	OPTIONAL

5 Detailed description 'collection' attribute syntax

Register the following attribute syntax, written in the style of section 4.1 Attribute Syntaxes of the IPP Model specification:

4.1.n 'collection'

A set of unordered attributes called member attributes, where each member attribute MAY be single-valued or multi-valued as specified for the collection attribute. The length of each collection value MUST be less than 1024 octets.

As in the attribute sets that are passed in an operation group, an IPP object MUST accept the attributes in a collection value in any order. The specification of an attribute whose attribute syntax is 'collection' MAY specify one or more member attributes that MUST be first in each collection value, in order to simplify processing, just as in the Operation attributes. If an attribute that is specified to be first is not in its required position, the IPP object MUST reject the operation and return the 'client-error-bad syntax' error status code. See [ipp-mod] Section 16.3.4.1.

No attribute SHALL occur more than once in a collection value. As in operation requests, if the same attribute does occur more than once in a collection value by error,

the IPP object MUST reject the operation and MUST return the 'client-error-bad syntax' error status code. See [ipp-mod] Section 16.3.4.3.

The specification of the attribute that uses the 'collection' attribute syntax specifies:

1. as with any attribute, whether the attribute is single-valued (attribute syntax = 'collection') or multi-valued (attribute-syntax = '1setOf collection').

2. for each member attribute in the collection value, whether the IPP object MUST support the attribute (REQUIRED) or MAY support the attribute (OPTIONAL).

3. for each member attribute in the collection value, whether the client MUST supply or MAY omit the member attribute in a request and whether the IPP object MUST supply or MAY omit the member attribute in a response.

4. for each member attribute permitted in the collection value, the completed specification of that member attribute is included or inferred by reference to the specification of that attribute elsewhere, including its keyword name, its attribute syntax, including '1setOf, if it is multi-valued, and the semantics of the values. The specification for a collection may include attributes that have already been defined for use by themselves and/or for use in other collections.

5. for each member attribute defined in the collection, whether that attribute may also be used separately by itself. For example, in the "job-notify" example, could the "job-notify-events" and "job-notify-recipients" attributes occur by themselves in a create operation, say, when the client is only specifying a single collection or must they always occur within a collection value.

6. for each member attribute defined in the collection, whether that attribute MAY occur anywhere in the collection value (the default case) or MUST be first or after some other attribute that MUST be first (must be explicitly specified).

A collection may contain another collection, i.e., may include a member attribute whose attribute syntax is, itself, a 'collection', if the specification of the (outer) collection attribute allows.

Additional attributes may be registered for use in a collection attribute.

Implementers MAY support additional private attributes in a collection value.

6 Encoding

This section shows the encoding for the alternative of representing a collection as a new attribute syntax. The new 'collection' attribute syntax will use the 0x34 tag value that has been reserved in the IPP/1.0: Protocol Specification [ipp-pro] for this purpose.

The following example is written in the style of the IPP/1.0 "Encoding and Transport" (nee "Protocol") document [ipp-pro]. In order to show a member attribute with multiple

values, the member attributes are specified as 1setOf, unlike the "job-notify" example b above (see section 4.2).

Octets	**Symbolic Value**	**Protocol field**	comments
0x34	collection type	value-tag	"job-notify" attribute
0x000a		name-length	
Job-notify	job-notify	Name	
0x0064		value-length	100 octets in 1st dict value
0x45	uri type	value-tag	"notify-recipients" attribute
0x0011		name-length	
notify-recipients	notify-recipients	Name	
0x0019		value-length	
ipp-tcpip-socket:port=700	ipp-tcpip-socket:port=700	Value	
0x44	keyword type	value-tag	"notify-event-groups" attribute
0x0013		name-length	
notify-event-groups	notify-event-groups	Name	
0x0b		value-length	
job-errors	job-errors group	Value	
0x44	keyword type	value-tag	start of 2nd job-notify-event-groups value
0x0000		name-length	0 length means next multiple value
0x000e		value-length	
job-completion	job-completion	Value	
0x34	collection-type	value-tag	start of 2nd collection value
0x0000		name-length	0 length mean next multiple value
0xnnnn	0xnnnn	value-length	nnnn octets in 2nd dict value
0x45	uri type	value-tag	"notify-recipients" attribute
0x0015		name-length	
notify-recipients	notify-recipients	Name	
0x000c		value-length	
mailto:smith	mailto:smith	Value	
...			nnnn octets of the next dict value

7 Rejected alternatives for a collection mechanism

This section lists the alternatives we considered for adding a new attribute syntax to represent a collection value.

1. Increase the maximum somewhat above the current maximum (1023), say, 2047 octets.

Reason for rejection: Not completely compatible with current parsers that have a fixed buffer size for entities of around 1023 octets, the current IPP data type maximum.

ISSUE: Is this rejection argument correct, because current parsers really do have a fixed buffer size? What about the case when the attribute syntax type is one that the implementation doesn't support and are going to ignore? They wouldn't need to return the value in the Ignored Attributes group, since we could clarify that a supported attribute that has an unsupported attribute syntax, is treated as an unsupported attribute, rather than as an unsupported value. Then the IPP object returns the attribute with the 'unsupported' out-of-band value, rather than the potentially longer than their buffer collection value. Or would it be a problem to current parsers to specify the maximum as 2047 octets for the 'collection' attribute syntax?

2. No maximum length for the new attribute syntax: 'collection'. If an IPP object supports collection it has to read a piece at a time. If it doesn't it has to be able to ignore an arbitrarily long data value. See the encoding example in the next section.

Reason for rejection: Not compatible with current parsers that have a fixed butter size for entities of around 1023 octets, the current IPP data type maximum.

ISSUE: Is this rejection argument correct, because current parsers have a fixed buffer size, even for attribute syntax types that they don't support and are going to ignore? They wouldn't need to return the value in the Ignored Attributes group, since we could clarify that a supported attribute that has an unsupported attribute syntax, is treated as an unsupported attribute, rather than as an unsupported value. Then the IPP object returns the attribute with the 'unsupported' out-of-band value, rather than the potentially longer than their buffer collection value.

3. Have a 1023 octet max length, continueCollection as a second attribute syntax and endCollection so that dictionaries can nest.

Reason for rejection: More complexity.

4. Have a 1023 octet max length but allow repeated instances of an attribute to append additional collection values.

Reason for rejection: Not the current procedure for duplicate attributes; the IPP Object is to return an error. See [ipp-mod] section 16.3.4.3.

5. Add a new group tag to represent a collection value somehow. Groups do NOT have lengths and existing parsers are supposed to ignore group tags they don't understand.

Reason for rejection: Not completely compatible with existing parsers.

6. Add an out-of-band value that indicates that this attribute was the beginning of a collection and add an attribute that marked the end of the collection value.

Reason for rejection: Not completely compatible with existing parsers. Existing parser would try to interpret the contents of the collection as regular attributes.

7. Extend the attribute naming mechanism to include a collection name and a collection index for use with multi-valued dictionaries. Use the colon (":") to separate component names. Thus if foo is a set of dictionaries, then "foo:1:x" is the name that accesses field x of the 2nd collection of attribute foo (indexing is 0 based). Leaving off the syntax after either colon, is interpreted as a wild card meaning all values with the prefix up to the colon.

Reason for rejection: Changing the naming is more of a change than is necessary with the current proposal, which simply adds an attribute syntax.

8. Use the semantics of parallel multi-valued attributes that we have in IPP/1.0, such as we already have for the "printer-uri-supported" and "uri-security-supported" Printer attributes, in order to achieve the effect of multi-valued dictionaries containing single values attributes. In order to represent the effect of a collection which contains attributes that are multi-valued, we only need to introduce the model semantics of: 1setOf 1setOf X as an attribute syntax.

Reason for rejection: Implementation experience with DPA [ISO-10175] parallel attributes has shown that it is too difficult for clients and servers to deal with parallel values. It is much better if the values in a collection value are all bound together. Also what if the number of values isn't the same in the parallel attributes?

9. Add a numeric instance number to the end of parallel attributes, i.e., "notify-method-supported-1".

Reason for rejection: Parallel attributes have proven as problematic in DPA implementations (see previous reason). Also we don't need the capability to be able to address a particular instance of a particular collection value.

8 IANA Considerations

This attribute syntax will be registered with IANA after the WG approves its specification according to the procedures for extension of the IPP/1.0 Model and Semantics [ipp-mod].

9 Internationalization Considerations

This attribute syntax by itself has no impact on internationalization. However, the member attributes that are subsequently defined for use in a collection may have internationalization considerations, as may any attribute.

10 Security Considerations

This attribute syntax causes no more security concerns than any attribute syntax. It is only the attributes that are subsequently defined to use this or any other attribute syntax that may have security concerns, depending on the semantics of the attribute.

11 References

[ipp-mod]
Isaacson, S., deBry, R., Hastings, T., Herriot, R., Powell, P., "Internet Printing Protocol/1.0: Model and Semantics" draft-ietf-ipp-mod-10.txt, June, 1998.

[ipp-not]
Isaacson, S., Martin, J., deBry, R., Hastings, T., "IPP Event Notifications (Very Short)" <ipp-notifications-very-short-980701.doc>, work in progress, July 1, 1998.

[ipp-pro]
Herriot, R., Butler, S., Moore, P., Tuner, R., "Internet Printing Protocol/1.0: Encoding and Transport", draft-ietf-ipp-pro-06.txt, June, 1998.

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