PWG DRAFT
File: ipp-collection-attr-syntax-981215.doc

Roger deBry
IBM Printing Company
T. Hastings
Xerox Corporation
R. Herriot
Sun Microsystems
December 15, 1998

Internet Printing Protocol/1.0: 'collection' attribute syntax

Status of this Memo:

This document is a PWG Working Draft. It specifies an OPTIONAL extension to the IPP/1.0 Model and Semantics document [ipp-mod]. 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. [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 an OPTIONAL 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. Each member attribute may be SINGLE-VALUED or MULTI-VALUED (1setOf).

Table of Contents

1 Problem Statement 2
2 Proposal for a 'collection' attribute syntax 2
2.1 Additional information provided in a collection specification 3
2.2 Extensions to collections 3
2.3 Default, supported, and ready collection attributes 4
2.4 Validation of collection attributes 4
3 Unsupported Values 6
4 Encoding 6
5 IANA Considerations 7
6 Internationalization Considerations 8
7 Security Considerations 8
8 References 8
9 APPENDIX A: Examples of collection usage 8
9.1 Example a: "printer-resolution" Job Template attribute 8
9.1.1 "printer-resolution-default" example 9
9.1.2 "printer-resolution-supported" example and validation of collections 9
9.2 Example b: "job-notify" Operation attribute 10
9.3 Example c: Start page fields supplied by the end-user 11
9.4 Example d: Postal mailing address 11
10 APPENDIX B: Rejected alternatives for a collection mechanism 12

1 Problem Statement

IPP currently lacks a mechanism for supporting attributes that contain several fields. It would be desirable to have a simple, general mechanism for representing multi-field attributes so that it is no longer necessary to create a new ad hoc attribute syntax for each new multi-field attribute, such as the 'resolution' attribute syntax for the "printer-resolution" attribute. Such a mechanism should allow some fields to be optional and others to be required. It would be useful for both private extensions and new registered attributes.

2 Proposal for a 'collection' attribute syntax

A value whose attribute syntax is 'collection' is a set of unordered attributes, each of which is called a member attribute. Each attribute in a collection has an attribute name that MUST be unique within the collection, but MAY be the same as the name of an attribute in another collection or in one of the attribute groups of an operation. Each member attribute is either single-valued or multi-valued. The length of a collection value is not limited by the model and semantics specification for the 'collection' attribute syntax, but may be limited by the encoding rules (see Section 4) However, the length of each member attribute MUST NOT exceed the limit of its attribute syntax.

Note: if a collection contains two or more member attributes with the same attribute name, the behavior of the receiver is undefined. The receiver could:

1. treat the entire collection as a bad value.
2. ignore all but the first occurrence of the member attribute.
3. ignore all but the last occurrence of the member attribute.

Note: The syntactic and semantic rules for a collection value are similar to the Job Template attribute group in an IPP request or a response. Both consist of an aggregation of attributes. However, a collection value, like any other value has a maximum length. A Job Template group is not a value and does not have a maximum length.

The general mechanism for collection values allows a collection value to consist of any set of member attributes. But when a collection value is associated with a particular attribute, the specification for that attribute MUST define the allowed member attributes of a collection for that attribute and related attributes.

According to existing rules, when a new attribute "xxx" is specified (for any attribute syntax), the specification must define the following:

1. attribute syntax of the attribute "xxx"
2. whether "xxx" is single-valued or multi-valued.

For a new Job-Template attribute "xxx", the specification must also define

3. whether there are associated Printer attributes: "xxx-supported" and "xxx-ready" and the attribute syntax and supported values for each. NOTE: for most attribute syntaxes, the attribute syntax of these two attributes is "1setOf" the attribute syntax of "xxx".

4. whether there are associated Printer attributes: "xxx-default". Note: the attribute syntax is the same as "xxx" and its value is one of the values of "xxx-supported".

2.1 Additional information provided in a collection specification

A specification of a new attribute "xxx" whose syntax type is 'collection' or '1setOf collection', MUST follow the four rules above. In addition, the specification must define the following information about each member attribute "yyy" of collection attribute "xxx":

1. the member attribute's keyword name ("yyy"),
2. the member attribute's ("yyy") attribute syntax, including '1setOf, if it is multi-valued. NOTE: the attribute syntax of "yyy" could be 'collection' or '1setOf collection'.
3. the complete semantic specification of the member attribute "yyy" Note: that its attribute name and semantics may be the same as an attribute in another collection or in an attribute group of an operation, and the description of "yyy" may reference the definition of its other use.
4. whether an implementation that supports attribute "xxx" MUST support the member attribute "yyy" (REQUIRED) or MAY support the member attribute (OPTIONAL).
5. whether the sender MUST supply or MAY omit the member attribute "yyy" in
    a) a request containing attribute "xxx",
    b) a response containing attribute "xxx",
6. what the default value of the member attribute "yyy" is if the create request includes "xxx" but does not include that member attribute "yyy".
7. what the supported values of a member attribute "yyy" are in a create request containing attribute "xxx".

2.2 Extensions to collections

After an attribute is registered that uses the 'collection' attribute syntax, additional member attributes may be registered subsequently for use in that collection attribute.

Furthermore, implementers MAY support additional private member attributes in such a collection attribute.

2.3 Default, supported, and ready collection attributes

If an attribute "xxx" has a collection value, and the Printer supports "xxx-default", "xxx-supported" or "xxx-ready" attributes:

1) "xxx-default" MUST be a "collection"

2) "xxx-supported" MUST be a "collection" or "1setOf collection".

3) "xxx-ready" MUST be a "collection" or "1setOf collection".

If an attribute "xxx" has a collection value, and the Printer contains an "xxx-default", then for every member attribute "yyy" of "xxx", "xxx-default" MUST either have a member attribute "yyy-default" or have an implementation default for "yyy".

If an attribute "xxx" has a collection value, and the Printer contains an "xxx-supported", then for every allowed member attribute "yyy" of "xxx", "xxx-supported" MUST have a member attribute "yyy-supported".

If an attribute "xxx" has a collection value, and the Printer contains an "xxx-ready", then for every allowed member attribute "yyy" of "xxx", "xxx-ready" MUST have a member attribute "yyy-ready".

If an attribute "xxx" has a collection value, and the Printer contains an "xxx-supported", then for every allowed member attribute "yyy" of "xxx", "xxx-supported" MAY have a member attribute "yyy-default" which specifies the value of "yyy" if it is omitted from "xxx".

If a collection contains the member attribute "attributes-required" and the collection is a value of an "xxx-supported" Printer attribute, then the value of the member attribute "attributes-required" is the set of member attributes names (1setOf keyword) required when a client sends an attributes "xxx".

2.4 Validation of collection attributes

The process of validating a Job-Template attribute "xxx" against a Printer attribute "xxx-supported" remains unchanged except for the addition of rules for determining "equality" of a collection value of "xxx" with one of the collection values of "xxx-supported".

The table below specifies the existing validation rules and adds a rule for collections. The following is the general validation algorithm (see section 3.2.1.2 in [ipp-mod]).

To validate the value U of Job-Template attribute "xxx" against the value V of Printer "xxx-supported", perform the following algorithm:

1. If U is multi-valued, validate each value X of U by performing the algorithm in Table 1~~Table 1~~ with each value X. Each validation is separate from the standpoint of returning unsupported values.

   Example: If U is "finishings" that the client supplies with 'staple', 'bind' values, then X takes on the successive values: 'staple', then 'bind'

2. If V is multi-valued, validate X against each Z of V by performing the algorithm in Table 1~~Table 1~~ with each value Z. If a value Z validates, the validation for the attribute value X succeeds. If it fails, the algorithm is applied to the next value Z of V. If there are no more values Z of V, validation fails.

   Example" If V is "sides-supported" with values: 'one-sided', 'two-sided-long', and 'two-sided-short', then Z takes on the successive values: 'one-sided', 'two-sided-long', and 'two-sided-short'. If the client supplies "sides" with 'two-sided-long', the first comparison fails ('one-sided' is not equal to 'two-sided-long'), the second comparison succeeds ('two-sided-long' is equal to 'two-sided-long"), and the third comparison ('two-sided-short' with 'two-sided-long') is not even performed.

3. If both U and V are single-valued, let X be U and Z be V and use the validation rules in Table 1~~Table 1~~.

Table 1 - Rules for validating single values X against Z

attribute syntax of X	attribute syntax of Z	validated if:
integer	rangeOfInteger	X is within the range of Z
uri	uriScheme	the uri scheme in X is equal to Z
collection	collection	the collection value Z supports X
any	boolean	the value of Z is TRUE
any	any	X and Z are of the same type and are equal.

A collection value Z MUST support a collection value X if the following is true:

1) for each member attribute "yyy" of X, Z contains a member attribute "yyy-supported" and the value of "yyy" validates against "yyy-supported".

2) If Z contains a member attribute "attributes-required", then for each value "y" of attribute "attributes-required" in Z, there is a member attribute "y" in X.

As an additional step in the validation of collections, a Printer MUST add a member attribute "yyy-default" to a value X if the value Z contains "yyy-default" and X does not contain "yyy".

NOTE: By having an "xxx-supported" attribute with more than one collection value, an implementation or administrator can indicate support for various combination of attributes, when not all combinations are supported. In addition, the defaults can differ for each supported value. See the example in section 9.1.

3 Unsupported Values

The rules for returning an unsupported collection attribute are an extension to the current rules.

1. If a collection is an unrecognized attribute syntax, its value is returned in the normal manner, except an implementation MAY return only first 1023 octets of the 'collection' value. If the value is truncated, the length returned MUST be the truncated length so that the response follows the syntax rules. This rule allows an Object which does not support a collection skip over the entire collection.

2. If a collection contains unrecognized and/or unsupported member attributes, the attribute returned in the Unsupported Group is a collection containing the unrecognized and/or unsupported member attributes. The unrecognized member attributes have an out-of-band value of unsupported. The unsupported member attributes have their unsupported values.

4 Encoding

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

Because the length field of a data type is encoded in two octets, the maximum length of a collection value MUST NOT exceed 32767 octets.

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 9.2).

Octets	**Symbolic Value**	**Protocol field**	comments
0x34	collection type	value-tag	"job-notify" attribute
0x000a		name-length	

Octets	**Symbolic Value**	**Protocol field**	comments
Job-notify	job-notify	Name	
0x0064		value-length	100 octets in 1st collection 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

5 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] and after IPP becomes a proposed IETF standard.

6 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.

7 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.

8 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-11.txt, November, 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-07.txt, November, 1998.

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

9 APPENDIX A: 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.

~~1.1~~9.1 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 to define resolution. If we did use the 'collection' attribute syntax for the "resolution", the attribute value would contain the following member 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 without being in a collection 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 member attributes:

Attribute name	syntax	member attribute
"resolution"	integer	MUST be present
"cross-feed-resolution"	integer	MAY be omitted
"resolution-units"	enum	MAY be omitted

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' }

9.1.1 "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-default" = '300' }

9.1.2 "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-supported" = '300', "attributes-required" = 'resolution' },
        { "resolution-supported" = '600', "attributes-required" = 'resolution',
          "cross-feed-resolution-supported" = '300' },
        { "resolution-supported" = '600', "attributes-required" = 'resolution' } }
```

Note: because there is no default indicated for "cross-feed-resolution", the default value is not fixed, but can be whatever the implementation wants, such as being the same as the value of the "resolution" supplied by the client.

If an implementation supported all combinations of 300 and 600 DPI, then it could more simply represent "printer-resolution-supported" as a single valued collection with multiple-values for each member attribute. It could also indicate that the "resolution" member attribute MUST be present and that the default value for the "resolution-units" attribute is 'dpi': what is default of cross-feed below?

```
"printer-resolution-supported" = {
        "resolution-supported" = '300', '600',
        "cross-feed-resolution-supported" = '300', '600',
        "attributes-required" = 'resolution'
        "resolution-units-default" = 'dpi' }
```

9.2 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	member attribute
"notify-event-group"	type2 keyword	MAY be omitted
"notify-recipient"	uri	MUST be present

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' }
```

The corresponding "job-notify-supported" might be:

```
    "job-notify-supported" = {
        "notify-event-group-supported" = 'job-errors', 'job-completion'
        "notify-recipient-supported" = 'mailto', 'ipp-tcpip-socket'
        "notify-recipient-required" = 'true' }
```

9.3 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

9.4 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

10 APPENDIX B: 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. Have a limit of 1023 octets for a collection value

Reason for rejection: For some uses of collection, it may be desirable to be able to supply more than 1023 octets worth of value. There is no need to limit the size of a collection. For those implementations that do support the 'collection' attribute syntax, they will parse each member attribute separately anyway, so that there is no need of a size limit on the collection value as a whole.

For those implementations that do not support the 'collection' attribute syntax, it is straightforward for an implementation to skip over arbitrary-sized (greater than 1023 octets) values. When returning unsupported attributes, only the out-of-band 'unsupported' value is returned in the Unsupported Attributes group, not the supplied value. Rejection of large (greater than 1023 octets) unsupported data types with unsupported attributes should be tested for in the next interoperability test session.

For those implementations that support an attribute, but do not support the 'collection' attribute syntax on that attribute need only return the first 1023 octets. Rejection of 'collection' attribute syntax on a supported attribute needs to be tested for in the next interoperability test session, including one with a value greater than 1023 octets.

2. Have a limit somewhat greater than 1023 octets, say, 2047 octets.

Reason for rejection: See above.

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 14.1.4.1.

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.

10. Define the collection as a subtype and set of values each containing a syntax type, a length and a value.

The subtype is an integer whose value is a registered subtype. The subtype specifies the order of values and the semantics and syntax type of each value. A value is omitted with a special out-of-band value called "omitted-value". Values that are sets of values are represented by a collection value whose subtype is "setOf".

Disadvantages: attributes with more than one value must be nested in a "setOf" collection and the sender must compute the length. Omitted attributes take up space, even when omitted.

Advantages: this solution is more compact. Collections intended for different uses are easily identified by their subtype. Otherwise, the "signature" of a collection is its member

433 names and possibly its member syntax types. With omission allowed for some members,
434 the signature gets more complicated.