Print Use Cases and Best Practice Solutions Using IPP

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
545	2012-11-20		SK

Contents

1 Introduction

The use case descriptions below represent stages or sub-tasks that users perform in the process of using a printer. Each of these below include a textual description as well as a series of workflow options for how it might be implemented using IPP. Each workflow option will be informally labeled according to its perceived quality, using the set of labels {"BAD", "MEH", "GOOD", "BETTER", "BEST"}, that are ordered from least desirable to most desirable.

2 Use Case Descriptions

2.1 Create a relationship with a particular printer

You can't print to a printer if you cannot establish a connection to it. Historically, connecting to a printer to establish a "relationship" with it meant identifying a printer and then creating a persistent local records and resources for that printer relationship with your system's print spooler. This was called a "print queue", and it involved binding drivers to create the relationships needed to communicate at the different levels, and then keeping record of that set of relationships so that it could be re-used at a later time. The set of printers or other devices the user's system might encounter was relatively small and fairly static.

More recent re-thinking of this relationship between client and printer has resulted in more "dynamic" relationship creation, where universal drivers can interrogate a device hosting a print service using a standardized protocol solution stack, and using that dynamically ascertain and update print service attributes. In this paradigm, a "persistent" print service record is more like a Web browser bookmark.

Both paradigms still require a method of identifying the target devices. That can be done using dynamic service discovery protocols where the services respond to discovery requests, or explicitly by name (host name or raw IPv4/IPv6 address).

2.1.1 Discover and select a printer via a discovery protocol

Discovery protocols are used to identify instances of print services or printers by searching the network for service types or device types. This helps the user by making it so that they don't need to do a physical survey of devices' addresses.

Regardless of the actual discovery protocol used, the APIs driving the protocols generally can be used in either a synchronous or asynchronous fashion. Unfortunately, many legacy software systems (as well as developers) are accustomed to the synchronous model, which is easily identified by the presence of a "refresh button". The synchronous model is not as user friendly as the asynchronous model, but it is somewhat easier to write programs in a synchronous way than an asynchronous way.

OPTIONS

- MEH:
  - Perform network discovery with a synchronous API
    - Show progress bar
    - Discovery.Start()
    - sleep(X) where X
    - Discovery.Stop()
  - Present the results of the discovery
  - "Refresh" button restarts process
    - Why this is bad:
      - list can be stale
      - it isn't "live"
      - "Reset" button is unnecessary and is a crutch
  - User selects a printer and presses "Continue" or equivalent
- BETTER:

  - Perform network discovery with an asynchronous API
    * Show List UI widget
    * Discovery.Start() with a callback
    * callback is called when discovery responses (add or remove) are received
  - User selects a printer and presses "Continue" or equivalent
    * Discovery.Stop()

2.1.2 Select a printer via user providing a DNS hostname or raw IPv4 / IPv6 address

In some cases a discovery protocol is either not adequate or unnecessary. Examples of when this use case is encountered include pre-published names or addresses, and also situations where the target device is not on the local link. (DNS-SD and WS-Discovery are generally used for link-local discovery, though wide-area variants as well as LDAP systems may also be used, but are frequently not for various reasons.)

For each of these options below, the assumption is that the client has received an address string, and wishes to connect to the

OPTIONS

- BAD:
  - Let each printer model make up its own path, and depend on some other protocol to get the resource path
    * IPP has no defined standard mechanism to enumerate the Printer objects' resource paths
- MEH:
  - IPP Get-Printer-Attributes with printer-uri set to a URI that was manually entered by the user
    * "ipp" URI scheme could be used to encode the hostname and the resource path
    * Having the user enter the URI exposes too many details to the user, including the detail about the fact that IPP is actually beiing used. Users need not be aware of which print protocol is being used.
- GOOD:
  - IPP Get-Printer-Attributes with printer-uri set to a well-known Printer resource path
    * "/ipp/print"
- BETTER:
  - IPP Get-Printer-Attributes with printer-uri set to "/"
  - examine the "printer-uri-supported" attribute; use the first URI in the list
  - IPP Get-Printer-Attributes with printer-uri set to first URI
- BEST:
  - IPP Get-Printers operation
    * YET TO BE DEFINED - COMING WITH SYSTEM CONTROL SERVICE
    * Is this really going to be better?

2.2 Get printer options to populate a print dialog (activity between doing "File→Print" and presenting the print dialog UI)

Once a printer has been identified, it is necessary for the print system to understand the capabilities that the printer device's print service provides. This includes what print job payload formats can be consumed by the print service, the available options and default choices, and so forth. It also includes other information about the device itself, such as its location.

OPTIONS:

- SAD:
  - Depend on a-priori knowledge about a particular model as a way of listing options for the model of device identified as the target
    - Model specific print drivers fall in this bucket
- MEH:
  - IPP Get-Printer-Attributes Operation
    - request includes no printer attributes; only operation attributes
    - reply will contain the job template attributes for all PDLs
  - Client guesses at what attributes may work or not work for a given PDL, or uses a-priori knowledge
- BETTER:
  - IPP Get-Printer-Attributes Operation
    - any specific attributes?
  - Process results; decide on a PDL
  - IPP Get-Printer-Attributes Operation
    - request includes the document-format attribute with value specifying the chosen PDL
    - reply will contain the job template attributes appropriately filtered ("colored") for that particular document-format

2.3 Check constraints between presented options

Printer features and options are presented typically in a print dialog. Some of these have states that have relationships with other options' states, where one cannot be in a particular state if another one is too. These are known as constraints, and they must be calculated any time the state of a control changes state. There are various ways that this can be done.

OPTIONS:

- BAD:
  - IPP Validate-Job
    - Every time a control is changed, the client sends IPP Validate-Job with attribute values corresponding to current state of controls
- MEH:
  - IPP Validate-Job
    - when "Print" button is pressed, confirms the job creation / submission will succeed (authentication, etc.)
    - client depends on this operation to perform constraints validation printer-side
- BETTER:
  - IPP Get-Printer-Attributes

    * device implements job-constraits-supported
    * device implements job-resolvers-supported
  - <Local processing of constraints>
  - IPP Validate-Job
    * when "Print" button is pressed, confirms the job creation / submission will succeed (authentication, etc.)
    * constraints validation already handled client-side

2.4 Submitting a Print Job

Once the user has decided on options, the print job is generated and ultimately made available to the printer in some fashion. There are several different ways that this may occur.

2.4.1 Submitting a print job "by value"

This is the classical way that a print job is sent from the client to the print service: a job is created, and the job information and job payload content are sent by the client to the print service.

OPTIONS:

- MEH:
  - IPP Print-Job
    * no pre-flighting
    * the printer may reject it but only after it has been transmitted in whole or in part.
    * better to check ticket and content types first.
- GOOD:
  - IPP Validate-Job
    * pre-flights the job by validating the job attributes and document type
  - IPP Print-Job
    * creates the job and sends the payload in one operation
    * however, the Job object's URI isn't usually known until the job transmission is complete
    * doesn't work well with flow-controlled (low-end) printers
- BETTER:
  - IPP Validate-Job
    * pre-flights the job by validating the job attributes and document type
  - IPP Create-Job
    * returns immediately with the job URI for monitoring and ticket processing status
    * if there is a problem then Create-Job may fail the same as Validate-Job would, but may not, which is why we do a Validate-Job first (so that there isn't a zombie job there)
  - IPP Send-Document
    * payload transmission is de-coupled from the creation of the job
    * multiple documents can be sent to build up a compound job
    * means that the client doesn't have to be prepared for an early HTTP response
    * allows the job URI to be learned before job payload is sent
    * MUST check to see if value of "multiple-document-jobs-supported" is "true", to see if it is OK to do multiple Send-Document operations to the same Job object.

2.4.2 Submitting a print job "by reference"

This is a slightly different way that a print job is sent from the client to the print service: a job is created and made available for retrieval by the print service, and when the print job the job information and job payload content are sent by the client to the print service.

OPTIONS:

- MEH:
  - IPP Print-URI
    - no pre-flighting
    - the printer may reject it but only after it has been transmitted in whole or in part.
    - better to check ticket and content types first.
- GOOD:
  - IPP Validate-Job
    - pre-flights the job by validating the job attributes and document type
  - IPP Print-URI
    - creates the job and sends a URL to where the payload can be retrieved in one operation
    - print service retrieves the payload file itself
    - however, the Job object's URI isn't usually known until the job transmission is complete
    - doesn't work well with flow-controlled (low-end) printers
- BETTER:
  - IPP Validate-Job
    - pre-flights the job by validating the job attributes and document type
  - IPP Create-Job
    - returns immediately with the job URI for monitoring and ticket processing status
    - if there is a problem then Create-Job will fail the same as Validate-Job would
  - IPP Send-URI
    - payload URI transmission is de-coupled from the creation of the job
    - means that the client doesn't have to be prepared for an early HTTP response
    - allows the job URI to be learned before job payload is sent

2.5 Monitoring print job status

While the print job is being processed, users may wish to know whether it is proceeding successfully, or whether there are conditions that they need to handle that are preventing processing from proceeding, such as a media jam, open covers, marking agents depleted, and so forth.

OPTIONS:

- MEH:
  - IPP Get-Printer-Attributes
    - monitor the value of the printer-state attribute
    - polling (lame)
- GOOD:

  - IPP Get-Job-Attributes
    - targets the specific job status
    - may be more precise
    - (need to see if job attributes will show printer issues (top cover open, etc.) that block a job as well)
    - polling (lame)

- BETTER:
  - IPP Create-Job-Subscriptions
    - asynchronous / long running queries for notifications that don't require polling

2.6 Canceling a print job

It may be that the user wants to terminate a job before it has been fully processed, for whatever reason. There are things that must be done to ensure that the state of the Job object has been decisively cleaned up by the client if the client is responsible for canceling the job. Clients leaving broken Job objects on the Print service is bad behavior.

There is also a dependency between the options below and how the job was submitted. If Print-Job is used, but the document payload is not completely transmitted, then is a Job object even created? Is this true in all cases? Alternately, if Create-Job / Send-Document is used, and the Cancel-Job is sent during the Send-Document operation submission, then

OPTIONS:

- BAD:
  - Client stop sending data and close the connection
    - *Problem:* The IPP Job Object may have been created and still exist, and be in an indeterminate state. It should be explicitly cleaned up by the client for best performance and correctness.
    - *Question:* Are there any realistic or theoretical conditions under which a Print-Job operation does NOT create a Job object?

- MEH:
  - Client stops sending chunks but doesn't close the connection because it is expecting an IPP operation reply.
  - IPP Cancel-Job operation request for the job via a second connection, which for some printers could result in a PDL interpreter hang because the last chunk sent didn't stop on a "statement" boundary
    - But if the client stops with a zero length chunk then the IPP stack will know that transport is complete
    - Need to define a "magic chunk" that operates kind of like an in-band inline Cancel-Job operation, to tell the PDL interpreter that no more job payload chunks?

- GOOD:
  - IPP Cancel-Job
  - Client stops sending chunks and closes the connection

- BETTER:
  - ???

2.7 Getting printer supplies status

Some administrative tasks, like checking consumables levels, are presented to end users in some cases, such as during print job status or in print dialogs. This is useful to end users and should be supported.

OPTIONS:

- MEH:
  - Don’t use IPP but use some proprietary protocol or platform-specific extension to IPP
    - The point is to use only IPP extensions based on open standards (i.e. PWG standard) and this violates that core principle
- GOOD:
  - IPP Get-Printer-Attributes
    - printer must implement JPS3 "printer-supply" attribute