USB Print Interface Class
IPP Protocol Specification

Revision 1.0

Revision 1.0 released on December 5, 2012

Authors:

Smith Kennedy, Hewlett-Packard Co.

Andrew R. Mitchell, Hewlett-Packard Co.

Contributors:

Robert McVay, Fresco Logic Inc.

Christopher Meyers, Fresco Logic, Inc.

Alan Berkema, Hewlett-Packard Co.

Jerry Thrasher, Lexmark

Justin Hutchings, Microsoft

Rahman Ismail, Intel Corp.

Scope of this Revision

Revision 1.0 of this document is the final ratified revision of this specification.

Table of Contents

1 Introduction

This IPP USB Specification describes an extension to the Universal Serial Bus Device Class Definition for Printing Devices previously developed by the USB Implementers Forum. This extension defines a standard for making the Internet Printing Protocol (IPP) available over a Universal Serial Bus (USB) Print class interface.

1.1 Scope

The information in this document fully describes IPP USB, a protocol extension to the existing Printer Class of USB devices. Included in this document are:

- IPP USB interface class, subclass, and protocol
- Interface descriptor extensions
- Definition of the Device Info Descriptor, a class-specific descriptor

Unless otherwise stated, devices implementing IPP USB interfaces MUST be a certified USB device, and comply fully with the following: the USB Common Class Specification; and the existing USB Print class specification version 1.1.

1.2 Purpose

The purpose of this document is to describe configuration, interface, and endpoint descriptors as well as a communications protocol for operating system, BIOS, and peripheral designers implementing support for USB printers that implement IPP USB as a new protocol type for the USB Print interface class. These definitions allow an operating system designer to design a single software package to support a given class or subclass of device. These definitions also provide a framework for designing the peripherals in each class or subclass.

Note This specification does not define Page Description Languages (PDL) or Printer Control Protocols (PCP). This specification defines a type of *interface* that is specifically intended to support the IPP PCP and existing PDLs.

1.3 Related Documents

Universal Serial Bus Revision 3.0 Specification:
http://www.usb.org/developers/docs/usb_30_spec_092911.zip

USB Common Class Specification:
http://www.usb.org/developers/devclass_docs/usbccs10.pdf

USB Print Class Specification:
http://www.usb.org/developers/devclass_docs/usbprint11.pdf

RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1

http://tools.ietf.org/html/rfc2616

RFC 2119: Key words for use in RFCs to Indicate Requirement Levels

http://tools.ietf.org/html/rfc2119

RFC 2910: Internet Printing Protocol/1.1: Encoding and Transport

http://tools.ietf.org/html/rfc2910

RFC 2911: Internet Printing Protocol/1.1: Model and Semantics

http://tools.ietf.org/html/rfc2911

RFC 3196, Internet Printing Protocol/1.1: Implementer's Guide

http://tools.ietf.org/html/rfc3196

PWG 5100.12: Internet Printing Protocol Version 2.0 Second Edition (IPP/2.0 SE)

ftp://ftp.pwg.org/pub/pwg/candidates/cs-ipp20-20110214-5100.12.pdf

PWG 5100.13 IPP: Job and Printer Extensions - Set 3

ftp://ftp.pwg.org/pub/pwg/candidates/cs-ippjobprinterext3v10-20120727-5100.13.pdf

1.4 Terms and Abbreviations

DNS-SD	DNS Service Discovery, a DNS based implementation of Zeroconf (Zero Configuration Networking)
IPP	Internet Printing Protocol
HTTP	Hyper Text Transfer Protocol
PWG	Printer Working Group
Status	Report or notification of the current state of the printer. Can include print job, or other activities that are occurring in the printer. Limited to the scope of what IPP defines for Printer Status or Job Status
USB	Universal Serial Bus
USB IF	USB Implementers Forum

2 Management Overview

The primary goal driving the creation of IPP USB is to raise the standard of service fidelity for printing and imaging devices on the USB, by leveraging the modern value propositions of IPP to the USB to provide a clearly defined full protocol solution stack.

Other USB interface classes have achieved a higher degree of universality by defining an interface with a level of fidelity that allows platform implementers to deploy a standard driver that works for all devices from all vendors. Examples of this are USB Mass Storage and USB HID. For devices that implement these interfaces, no additional model-specific drivers are needed for full-featured use in most cases.

The Internet Printing Protocol (IPP) can be used for querying the device for its capabilities and options to be presented to the user, to send jobs and their associated intent and metadata from host to device or device to host, and to query the device for status and other ancillary but related information. IPP is an IETF open standard protocol that replaces numerous vendor-proprietary protocols, which is key to simplifying and standardizing the protocol stack solution for printing and imaging.

IPP USB is intended to allow USB connected devices to achieve functional parity with network-connected devices using IPP.

3 Functional Characteristics

3.1 Interfaces

An IPP USB interface is a USB Print class interface (class 7) that implements protocol level 4 as one of its interface alternates. As with the earlier Print class interface protocol levels, a Print class interface implementing IPP USB (protocol level 4) provides 2 endpoints: 1 Bulk IN and 1 Bulk OUT.

When host and device concur on using protocol level 4 for that interface, through the device providing an interface alternate listing protocol 4 and the host selecting that alternate, they are agreeing with each other to use that interface in IPP mode, meaning that each side will expect to receive and interpret only IPP messages, and will only be sending IPP messages. This is similar to how protocol level 3 (1284.4) works: when the device provides an alternate interface with protocol level 3, and the host chooses that alternate, both sides agree that all communications on that interface's endpoints will be 1284.4 between host and device resident 1284.4 protocol stacks.

3.2 Interface Set

Devices MUST implement at least 2 IPP USB capable Print class USB interfaces to comply with this specification, but MAY implement more than 2. This is required to provide a baseline service set that can handle cases where one of the interfaces is already in use. Devices MAY implement as many additional IPP USB interfaces as are deemed necessary to avoid contention in accessing the facilities provided by that device.

All IPP USB capable Print class interfaces provided by a device MUST be functionally equal from an IPP operation or HTTP perspective. In other words, any IPP operation or resource path that is valid over one IPP USB interface MUST be reachable via any and

all of the IPP USB interfaces. All the IPP USB interfaces provided by a printer must connect to the same IPP Server in the printer.

All IPP USB interfaces SHOULD be considered functionally equal by the USB host operating system and software running on it, from an IPP operation and HTTP resource availability perspective.

This does not mean that all IPP USB capable interfaces must be interchangeable and identical at the USB level. It is perfectly legitimate (and may be common) to have a device that implements an IPP USB capable Print class interface that also implements other protocol levels via interface alternates (such as protocol 2 or 1 for backward compatibility with USB host operating system environments that don't provide IPP USB support), while all the other IPP USB capable Print class interfaces provide only protocol 4 as interface alternate 0, and no other interface alternates.

Some devices may provide multiple functions (i.e., print, scan, fax, configuration) via these interfaces, whether implemented using IPP or other HTTP based protocols. These functions may be usable concurrently. In such cases multiple clients may be trying to use these interfaces at the same time. For this reason, and to minimize the chance of connection contention, such a device SHOULD implement a number of IPP USB capable Print class interfaces equal to or greater than one more than the number of functions provided over an IPP USB interface on that device, to provide a "spare" interface for out-of-band status and other operations. As an example, if a device implemented 3 functions that were to be accessed via IPP USB interfaces then the device should provide 4 IPP USB interfaces.

4 Descriptors

Unless otherwise stated, for full descriptor definitions see the USB Device Class Definition for Printing Devices, the USB Common Class Specification or the USB 2.0 specification set.

4.1 Interface Descriptor

IPP USB is an extension of the existing USB Print interface class. Thus the interface descriptor will be exactly the same as what is currently defined in the USB Print Interface Class specification, with the one change that the value for bInterfaceProtocol can now have the value 4, indicating IPP USB. For each USB Print class interface that supports IPP USB, one of that interface's alternate configuration descriptors MUST specify the value 4 for the bInterfaceProtocol field to indicate this support.

Offset	Field	Size (Bytes)	Value	Description
0	bLength	1	09h	Size of this descriptor, in bytes.
1	bDescriptorType	1	04h	INTERFACE descriptor type.
2	bInterfaceNumber	1	00h	Zero-based value identifying the number of this interface.
3	bAlternateSetting	1	??h	Value used to select an alternate interface.
4	bNumEndpoints	1	??h	Number of endpoints used by this descriptor. This is 01h or 02h for printers
5	bInterfaceClass	1	07h	Base class for printers.
6	bInterfaceSubClass	1	01h	The subclass codes for Printer devices are: 01 Printers
7	bInterfaceProtocol	1	??h	Printer Interface Type: 00 Reserved, undefined. 01 Unidirectional interface. 02 Bi-directional interface. 03 1284.4 compatible bi-directional interface. 04 IPP USB interface 05-FEh Reserved for future use. FFh Vendor-specific printers do not use class-specific protocols.
8	iInterface	1	??h	Index to string that describes this interface.

(Note: the "Universal Serial Bus Device Class Definition for Printing Devices" specification version 1.1 mistakenly lists 'bInterfaceSubClass' as 'iInterfaceSubClass' - that typo has been fixed in the interface descriptor description table above.)

4.2 Endpoint Descriptor

The Endpoint Descriptor for all IPP USB interfaces is defined as follows:

Attributes	In/Out	Type	Description
Bulk OUT	Out	REQUIRED	Bulk OUT Endpoint
Bulk IN	In	REQUIRED	Bulk IN Endpoint

4.3 Device Info Descriptor: A Class Specific Descriptor

Each USB interface class may define one or more class-specific descriptors and specify whether those descriptors are mandatory or optional. These class-specific descriptors do not replace the other standard USB descriptors, but rather provide additional information specific to that interface class.

A device implementing IPP USB MUST provide the Device Info Descriptor described below, and it must be structured in the way that is described in this section.

The table below describes the structure of the Device Info Descriptor. The overall structure of the Device Info Descriptor contains 3 distinct areas. First, there are a set of fields that describe the size and structure of the entire descriptor. These fields are bLength, bDescriptorType, bcdReleaseNumber, and bNumDescriptors. Next are the mandatory capabilities descriptors that MUST be present. Lastly, there may be vendor-specific capabilities descriptors. These 3 areas have been color coded in the table at the end of this section, for clarity.

The value of bNumDescriptors specifies the total number of capabilities descriptors present in the Device Info Descriptor. The minimum allowed value for this field, as of this specification, is 1, since the IPP Basic Capabilities descriptor is the only mandatory one currently defined. In future versions of this specification this may change.

Each capabilities descriptor, regardless of its type, identifies its type and length with two fields: bCapabilitiesType, and bCapabilitiesLength. Legitimate values for the "bCapabilitiesType" fields are as follows:

- 0x00: IPP Basic Capabilities descriptor type
- 0x01 - 0x1F: Reserved for future mandatory capabilities descriptors
- 0x20 - 0xFF: Optional / vendor-specific

The IPP Basic Capabilities descriptor contains information about general use and capabilities of the IPP USB interfaces and negotiating the connection itself. However, specific capabilities information for IPP functions, such as print capabilities or PDLs supported, etc. are not available via any of the standard capabilities descriptors within the Device Info Descriptor, and must be acquired by software on the host via IPP operations sent using one of the IPP USB interfaces.

Little endian byte ordering and bit numbering is used in this descriptor, for consistency with the conventions used on the USB. All bitfields start with the Least Significant Bit (LSB) in location 1 and work through the Most Significant Bit (MSB) in the last location (8,16 etc. depending on the length in bytes of the bitfield). Vendor Class Specific Descriptors should also follow this byte ordering and bit numbering convention.

Offset	Field	Size (Bytes)	Value	Description
0	bLength	1	0ah	Size of the descriptor in bytes (always >= 10)
1	bDescriptorType	1	21h	Descriptor type code (assigned by USB-IF)
2	bcdReleaseNumber	1	01h	Print Class Specification release major version number
3	bNumDescriptors	1	01h	Number of IPP capabilities descriptors to follow (always >= 1)
4	bCapabilitiesType	1	00h	IPP Basic Capabilities Descriptor
5	bCapabilitiesLength	1	04h	Length of capabilities descriptor
6	wBasicCapabilities	2		IPP Basic Capabilities Bitfield: Bit 1: Print Bit 2: Scan Bit 3: Fax Bit 4: Other (Vendor Specific) Bit 5: Any HTTP/1.1 over USB Bits 6-7: Authentication: 00 - None 01 - Username/Password 10 - Reserved 11 - negotiate Bits 8-16: Reserved (all bits set to 0)
8	iIPPVersionsSupported	1	00h	Index to string containing a comma delimited list of IPP versions supported by the device's IPP server. If value is 00h, IPP operations should be used to acquire this from the device's IPP server.
9	iIPPPrinterUUID	1	00h	Index to string containing the "printer-uuid" IPP attribute value for the IPP Printer object at /ipp/print. If value is 00h, IPP operations must be used to acquire this from the device's IPP server.
[10]	[bCapabilitiesType]	[1]	00h	Type of first optional capabilities descriptor (value chosen from the range 0x20 - 0xFF)
[11]	[bCapabilitiesLength]	[1]	??h	Length of capabilities descriptor
[12-?]	[Various fields...]	??		Whatever fields and values are relevant to this optional, vendor-specific capabilities descriptor

5 Requests

This specification defines no new Class-specific device requests and will preserve those that are currently defined for the USB Print interface class.

However, there are recommendations as to the keys and values in the 1284 Device ID string value returned by the GET_DEVICE_ID (bRequest = 0) class-specific device request type, to ensure proper behavior by USB host systems' print driver matching subsystems. These will be described in a separate implementer's guide, since such recommendations may change over time as details are discovered. Keeping them separate will help to reduce the need for updates to this specification itself.

6 Operational Model

6.1 HTTP Connections

Since IPP is based on HTTP and is a specialized usage of HTTP, and since many printers also have an Embedded Web Server (EWS) for device configuration, as well as other services that may be provided via HTTP, the device MAY allow general purpose HTTP on its IPP USB interfaces if they are able to appropriately disambiguate IPP from general-purpose HTTP traffic.

To allow the host to determine if the IPP USB interfaces support this extended HTTP functionality, a client can consult the state of the "Any HTTP/1.1 over USB" bit in the wBasicCapabilities field of the Device Info Descriptor (defined above). Devices that support this capability MAY set this bit to allow clients to know that this is supported.

The USB Device MUST implement and support only HTTP/1.1 or later, and MUST NOT support HTTP/1.0. HTTP/1.1 is required because HTTP/1.0 requires the server to close the connection, but USB devices cannot do this. Therefore, it will always be the host's job to close the connection.

If the USB Device receives an HTTP/1.0 request, it MUST reject the request by replying with an HTTP status code 505 "HTTP Version Not Supported".

Also, since all IPP USB interfaces MUST be functionally identical and interchangeable as far as the basic capabilities they provide, the IPP operations supported and the IPP objects to which they provide access, similarly any resource accessible via HTTP on any one IPP USB interface MUST be available on all IPP USB interfaces. Put another way, for a given resource at a particular path, that resource must be accessible via HTTP on all IPP USB interfaces if it is available on one of them. Host software MUST not have to figure out which resources are available via which interfaces.

6.2 HTTP Headers

Since there is no network connection, no DNS hostnames or IP addresses, and no TCP port numbers associated with the USB connection, the requirement of the HTTP Host field is addressed by requiring that the value of this header MUST be “localhost”. This means the following HTTP header line:

```
Host: localhost
```

MUST be in all HTTP requests sent over the USB interface. To be clear, HTTP/1.1 requires the presence of this header field in all HTTP requests, and this specification mandates the value of the hostname portion of the header value. If the USB Host chooses to include a port number, it is free to do so but it MUST have no influence on the choice of a particular IPP USB interface or how the IPP USB interface connections are managed.

The USB device's IPP server / HTTP server MUST use the value of the "Host" HTTP header received in HTTP requests for the value in the "Server" header of its corresponding HTTP replies. The USB device's IPP server / HTTP server MUST use the value of the "Host" HTTP header received in HTTP requests (mandatory header for HTTP/1.1), including both the hostname and port number, in all self-referencing absolute URLs in any IPP payloads (attribute values, etc.) and in in any content returned in the HTTP replies.

6.3 HTTP and IPP Paths

IPP operations require a static path to specify the IPP Printer Object maintained by the IPP server. All IPP objects will reside at a base path "/ipp". The path for the single Printer object provided by the device MUST be:

```
/ipp/print
```

The name is "print" because that is the function or service that the IPP object provides. This is equivalent to the following rp key listing in the DNS-SD TXT record:

```
rp=ipp/print
```

If the device specifies support for general purpose HTTP via the "Any HTTP/1.1 over USB" bit in the Device Info Descriptor's wBasicCapabilities bitfield, it is assumed that the default function of this is to access the embedded web server in the device, to allow device configuration and administration. For consistency with conventional web sites, the path for general purpose HTTP MUST be the "root" path served by that web server:

```
/
```

If, for example, the device has the "Any HTTP/1.1 over USB" bit set in the wBasicCapabilities descriptor field, and software on the host wanted to get the default HTML content for the device, it would open one of the IPP USB interfaces' endpoints, and write the following request to the Bulk OUT endpoint:

```
GET / HTTP/1.1\r\n
Host: localhost\r\n
\r\n
```

The software on the host would then begin monitoring the Bulk IN for a reply from the device in response to the above request. The specifics of the reply content are beyond the scope of this document, but a host performing a "GET / HTTP/1.1" should eventually be able to get to the default content in the device, assuming all authentication and encryption requirements have been satisfied. The point is that the "/" path is a legitimate path alias for the default content provided by the device's web server.

6.4 IPP Printer URI

Given the definition of the queue path equivalent, a hostname substitute, and the HTTP Host header field, a full IPP printer-uri can be built for clients to use when creating their IPP requests. Putting all the pieces together yields the following URI:

```
ipp://localhost/ipp/print
```

6.5 Icon File Path

If the device advertises support for HTTP transactions, the device MAY provide an icon image representation of itself. If the USB device does provide an icon file, the path to the icon file MUST be:

```
/icon.png
```

This path is case-sensitive. The icon file MUST be in Portable Network Graphics (PNG) format, MUST have size dimensions of 128x128 pixels, and MUST have an alpha channel to mask out the background. Additional representations, if present, MAY be determined through IPP operations (which operations are outside the scope of this specification).

7 Class Interactions

As mentioned earlier in this document, this specification does not itself define its own USB interface class. Rather, it extends the existing USB Print interface class specification by defining a new protocol. All attributes of any kind about an IPP USB interface that are not defined by this specification are, or should be, defined by the USB Print interface class specifications or one of the other specifications listed in the References section.

8 Conformance Requirements

This section summarizes the Conformance Requirements detailed in the definitions in this document for USB Hosts implementing IPP USB driver support and USB Printers implementing IPP USB interfaces.

8.1 Conformance Requirements for this Specification for USB Devices

USB devices conforming to this specification:

1. MUST be a certified USB device.
2. MUST conform fully to the USB Common Class Specification 1.0.
3. MUST conform to the latest existing USB Print Interface class specification (Universal Serial Bus Device Class Definition for Printing Devices, Version 1.1, January 2000).
4. MUST implement at least 2 Print class interfaces, each of which:
    a. MUST conform to the aforementioned USB Print Interface class specification.
    b. MUST advertise protocol 4 as one of that USB interface's alternates via USB interface descriptors.

c. MUST provide a Device Info class-specific descriptor as defined in Section 4.4 of this specification.

5. MUST implement a single IPP service with the following characteristics:
    a. MUST implement at least IPP version 1.1
    b. MUST provide uniform consistent access to all IPP objects (Printer, Job, Subscription) via each and any of the IPP USB interfaces the device implements, such that any IPP operation, object or attribute available via any one IPP USB interface MUST BE available on all the other IPP USB interfaces as well.
    c. MUST implement an IPP Printer Object at the resource path "/ipp/print"
    d. MAY provide an icon file; if so, it MUST have the following characteristics:
        i. MUST be found at resource path "/icon.png"
        ii. MUST be 128 pixels wide and 128 pixels high
        iii. MUST have an alpha channel
    e. MUST use the value of the "Host" HTTP header received in HTTP requests (mandatory header for HTTP/1.1) for the value in the "Server" header of its corresponding HTTP replies
    f. MUST use the value of the "Host" HTTP header received in HTTP requests (mandatory header for HTTP/1.1), including both the hostname and port number, in all self-referencing absolute URLs in any IPP payloads (attribute values, etc.)
6. MAY implement an HTTP service that can accept non-IPP HTTP requests and respond with conventional HTTP transported content, such as HTML, XML, etc.
    a. MUST set the "Any HTTP/1.1 over USB" bit in the Device Info descriptor to tell the host that the device supports this.
    b. MUST use "/" as its document root.
    c. MUST provide uniform consistent access to all available resources via each and any of the IPP USB interfaces the device implements, such that a resource available via HTTP over one IPP USB interface MUST BE available on all the other IPP USB interfaces as well.
    d. MUST use the value of the "Host" HTTP header received in HTTP requests (mandatory header for HTTP/1.1) for the value in the "Server" header of its corresponding HTTP replies
    e. MUST use the value of the "Host" HTTP header received in HTTP requests (mandatory header for HTTP/1.1), including both the hostname and port number, in all self-referencing absolute URLs in any content returned in the HTTP replies
7. Each Print class interface that is configured to use the interface alternate listing protocol 4 for the interface descriptor's bInterfaceProtocol field:

a. MUST connect to the printer's single IPP service:
    i. Data received by the USB device from the USB Host over the pipe corresponding to the USB host's Bulk Out endpoint for that interface MUST be IPP client operation requests that conform with RFC 2910. If data received by the USB device from the USB Host is not part of a legitimate IPP client operation request that conform with RFC 2910, the device MUST respond with appropriate IPP or HTTP server error responses (since IPP's transport is defined as HTTP/1.1), depending on the nature of the problem detected by the USB device with the request (either IPP or HTTP).
    ii. Data sent by the USB device to the USB Host over the pipe corresponding to the USB host's Bulk In endpoint for that interface MUST be IPP server operation responses that conform with RFC 2910.
    iii. For all of the USB device's Print class interfaces supporting protocol 4 as one of that interface's alternates, the scope of usability in terms of access to the device's IPP Objects, operations, and attributes, MUST be identical.
b. MAY also connect to an HTTP service if the "Any HTTP/1.1 over USB" bit is set in the Device Info class-specific descriptor:
    i. Data received by the USB device from the USB Host over the pipe corresponding to the USB host's Bulk Out endpoint for that interface MUST be HTTP requests that conform with RFC 2616, but MAY be IPP operation requests that conform with RFC 2910.
    ii. Data sent by the USB device to the USB Host over the pipe corresponding to the USB host's Bulk In endpoint for that interface MUST be HTTP server responses to received requests, and these responses MUST conform with RFC 2616.
    iii. The USB device MUST be able to route the request to either the printer's HTTP or IPP server (if they are separate entities within the printer) based on whether the request conforms to RFC 2910 or not.
c. MUST reject any HTTP/1.0 request with an HTTP 505 "HTTP Version Not Supported" status response

8.2 Conformance Requirements for this Specification for USB Hosts

In order for a USB Host to claim conformance to this specification, it MUST support the following:

1. MAY implement USB Print interface class driver or drivers that can match and use an IPP USB interface (class 7, subclass 1, protocol 4).
2. MUST NOT implement drivers that assume that protocol 4 is the alternate #0 for a given interface.
3. If a matching driver suite chooses protocol 4, the host MUST create only 1 print queue for the IPP Printer object hosted by the device, not one queue for each Print class USB interface.
4. MUST send data compliant with RFC 2616 over any IPP USB interface's bulk out pipe
5. MUST send data compliant with RFC 2910 over any IPP USB interface that does not have the "Any HTTP/1.1 over USB" bit set in the Device Info descriptor

Appendix A: Use Cases (Informative)

The following are examples of ways in which users could take advantage of an IPP USB implementation.

A.1 Walk up and print

The user connects their printer to their computer or other device that can operate as a USB Host.

The USB Host has an operating-system standard IPP USB interface class driver. The USB Host also has a universal or generic IPP print driver that is capable of interrogating the printer for its capabilities and generating one of the job formats (job Page Description Language) that the IPP USB device can consume and process to print the job correctly.

The USB hub driver on the USB Host recognizes the USB connection event when a device is connected to it. The USB hub driver interrogates the device to enumerate its descriptors, and the driver matching subsystem begins its standard USB driver matching process to match drivers to that model of device or its interfaces. The driver matching subsystem matches the operating system's standard USB Print interface class driver because it matches the Print interface class (class 7), and no model-specific driver takes precedence over that Print interface class driver.

Once loaded, the USB Print interface class driver considers the set of interface protocols supported by that interface on all of the interface's alternates. If the Print interface class driver finds that there are 2+ Print class interfaces, each of which has an alternate with bInterfaceProtocol 4, the Print interface class driver may then enumerate the print function as IPP Print in order to later look for a match for an IPP print driver. Subsequent matches of a Print interface class driver should check to see if a queue had been previously created, and instead provision the other interfaces with drivers, etc. that allow user space software to use these IPP interfaces for other functions on demand, such as for out-of-band status.

If there are not 2+ Print class interfaces that provide protocol 4 as an interface alternate, then it enumerates the function as a conventional Print function. In this case, the "legacy" USB hot-plug experience should be presented to the user, which might include, for instance, prompting the user to install a traditional model-specific driver, or it may ignore the USB hot-plug event if no matching driver is found. (The details of this are specific to the USB host environment and beyond the scope of this specification.)

Appendix B: Data Flow (Informative)

Below are diagrams and descriptions of example connection scenarios to help the reader visualize the value proposition of IPP USB more clearly.

B.1 Multiple IPP USB Interfaces, Only One in Use

Figure 1 represents a scenario where a device implements the required 2 or more IPP USB interfaces, but only one is in use:

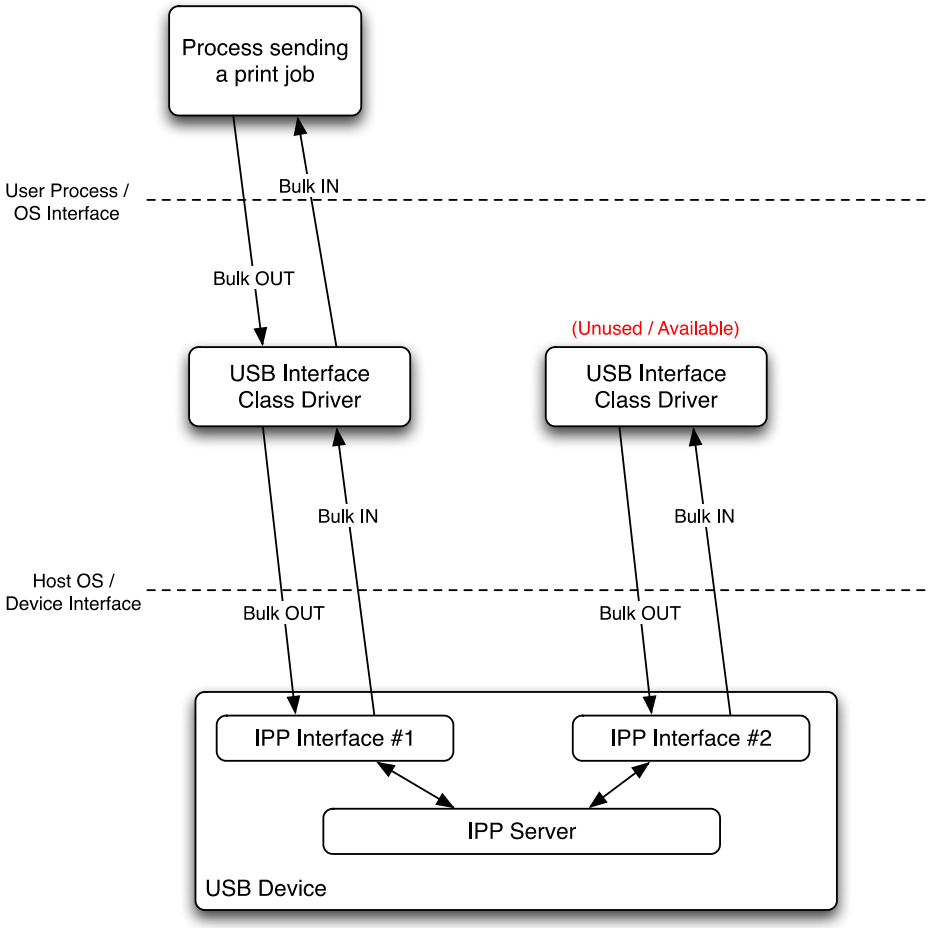

Figure 1: One IPP USB Interface in Use, One Unused

As Figure 1 illustrates, each IPP USB interface provides two endpoints, Bulk IN and Bulk OUT. A process on the host using this interface writes HTTP requests to the Bulk OUT endpoint, and reads HTTP responses from the Bulk IN endpoint.

The device must implement at least two separate IPP USB interfaces, in order to provide IPP access to other processes that might want to monitor job or device status to do so out-of-band of the interface being used for job transmission. This will be covered in Section B.2.

B.2 Multiple IPP USB Interfaces, All In Use

Figure 2 presents a scenario where a device implements the minimum required 2 IPP USB interfaces, and both are in use - one by a process sending a job to the device, and the other process monitoring the device or job status:

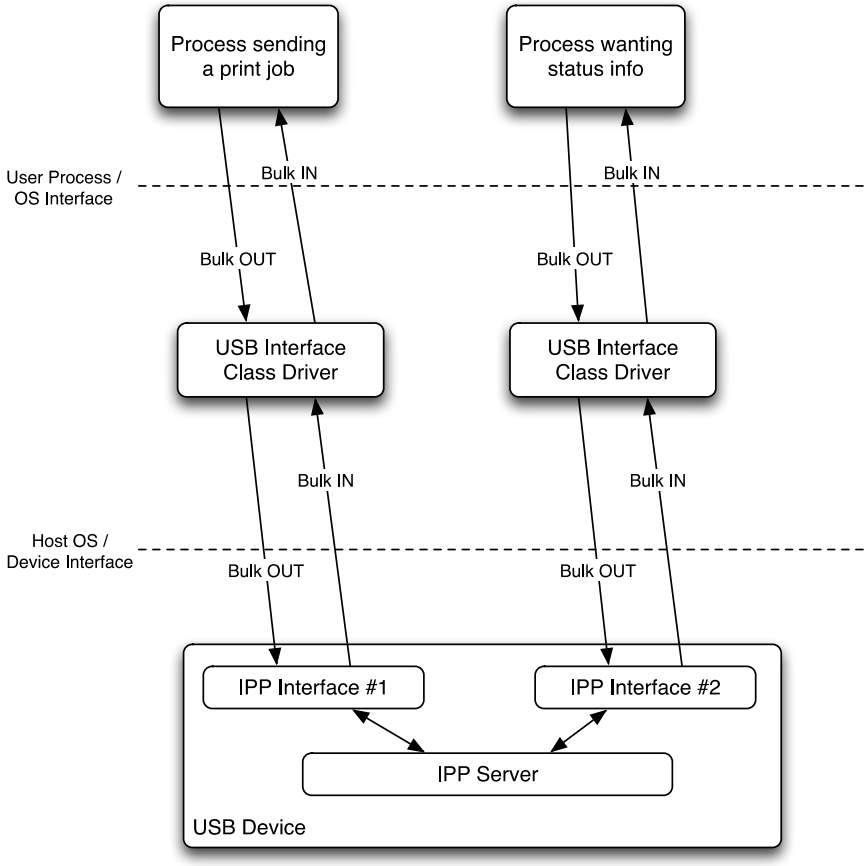

Figure 2: Two IPP USB Interfaces, Both in Use

To ensure that job transmission does not prevent other processes from acquiring device or job status or performing other IPP operations, a second IPP USB Interface is required. Both interfaces are functionally identical and interchangeable from an IPP capability perspective, as mandated in Section 3.2. As such, it could be that the process sending a print job might connect through the interface on the right, and the process wanting status info would connect through the interface on the left. Doing so should have no discernable difference from the perspective of the processes at the top of the diagram.

B.3 Handling Contention with IPP USB Interfaces

A process performing IPP operations where the reply is received immediately, such as Get-Printer-Attributes, Get-Job-Attributes or Cancel-Job, should close the USB interface once the IPP operation reply has been received, to allow other processes an opportunity to also use that interface. Such a conflict is illustrated in Figure 3:

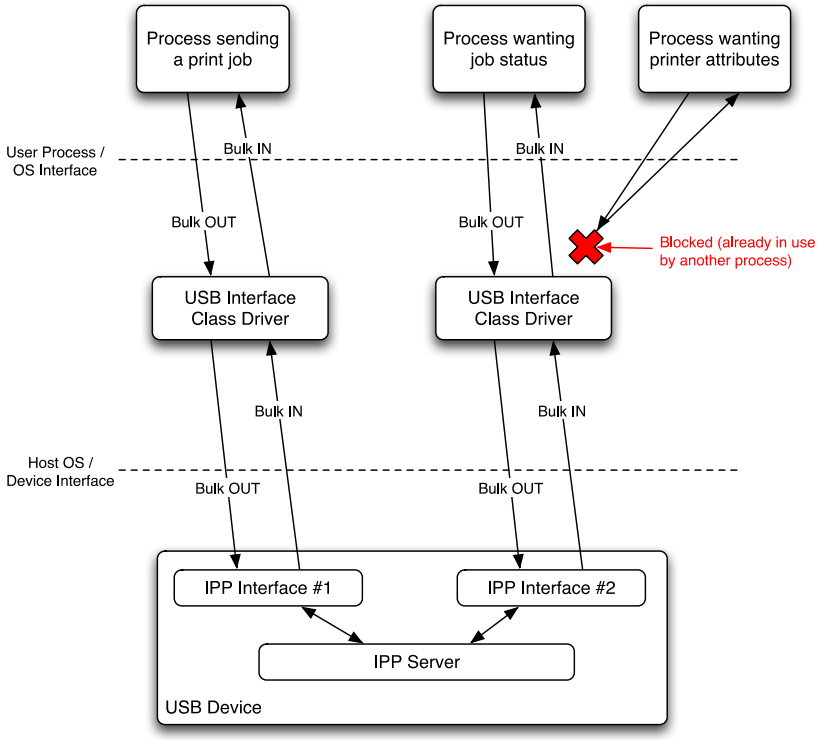

Figure 3: Two IPP USB Interfaces with interface contention

As shown by Figure 3, a process that selfishly assuming it is the only one with a need to contact a device using a given IPP USB interface can lead to starvation of other processes that may also need access to the same IPP USB device. To prevent this scenario, processes need to limit the amount of time they have an IPP USB interface open to the time required to perform a single request and receive a reply.

However, it is possible that processes are not well mannered. One possible way to prevent ill-mannered processes causing access starvation might be for the USB host operating system to implement an arbitration layer that would act as a “proxy” IPP Printer Object, that could cache information such as Job Template attributes from the Printer Object, notifications received via IPP Subscriptions, etc. This would allow processes to get device attribute information and/or create subscriptions for status events with the status notification cache service, which will send status notifications when events happen, without the need for the process to actually contact the device. This scheme might look like what is shown in Figure 4:

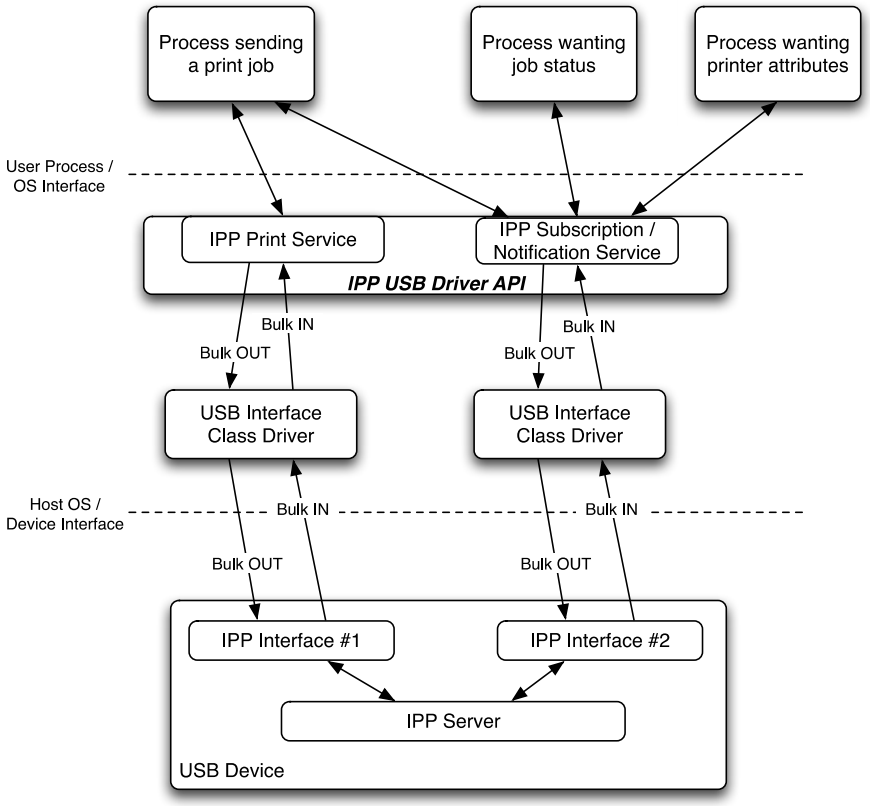

Figure 4: Two IPP USB interfaces with Status Service

Figure 4 shows status data going to both the Process sending the print job and other, different processes. Status could also be provided only to the process sending the job or only to a different process.

Appendix C: IPP Data Flow Sequence (Informative)

C.1 Simple IPP Request / Reply Sequence Diagram

Figure 5 below presents how the connection and data flow would proceed in the process of using the interface to perform one IPP operation, and receive the reply.

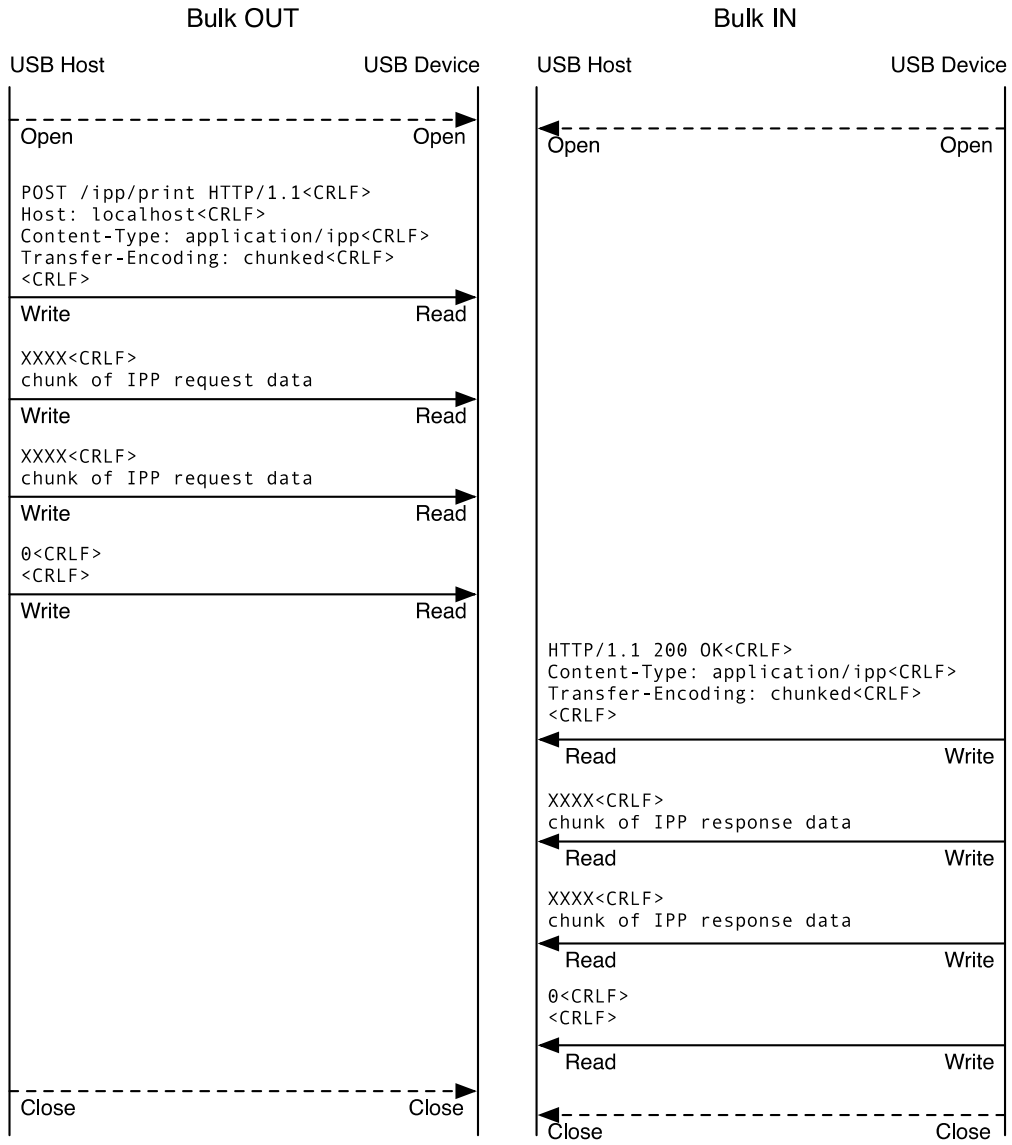

Figure 5: Communications sequence diagram for the bulk pipes of an IPP USB interface

USB Hosts write data on the pipe corresponding to the Bulk OUT Endpoint, and read data from the pipe corresponding to the Bulk IN Endpoint.

USB Devices read data from the pipe corresponding to the host's Bulk OUT Endpoint and write data on the Bulk IN Endpoint.

USB Hosts open both Bulk OUT and Bulk IN at the same time and start looking for data immediately on Bulk IN for cases where the USB Device sends a reply before the entire

request is sent. This is common in cases where the device is upgrading the HTTP request to HTTPS.

C.2 USB Host and Device Interaction Sequence Diagram 1

This sequence diagram shows 2 processes engaged in overlapping print dialog and job submission, where the host implements caching of the results of the Get-Printer-Attributes IPP operation.

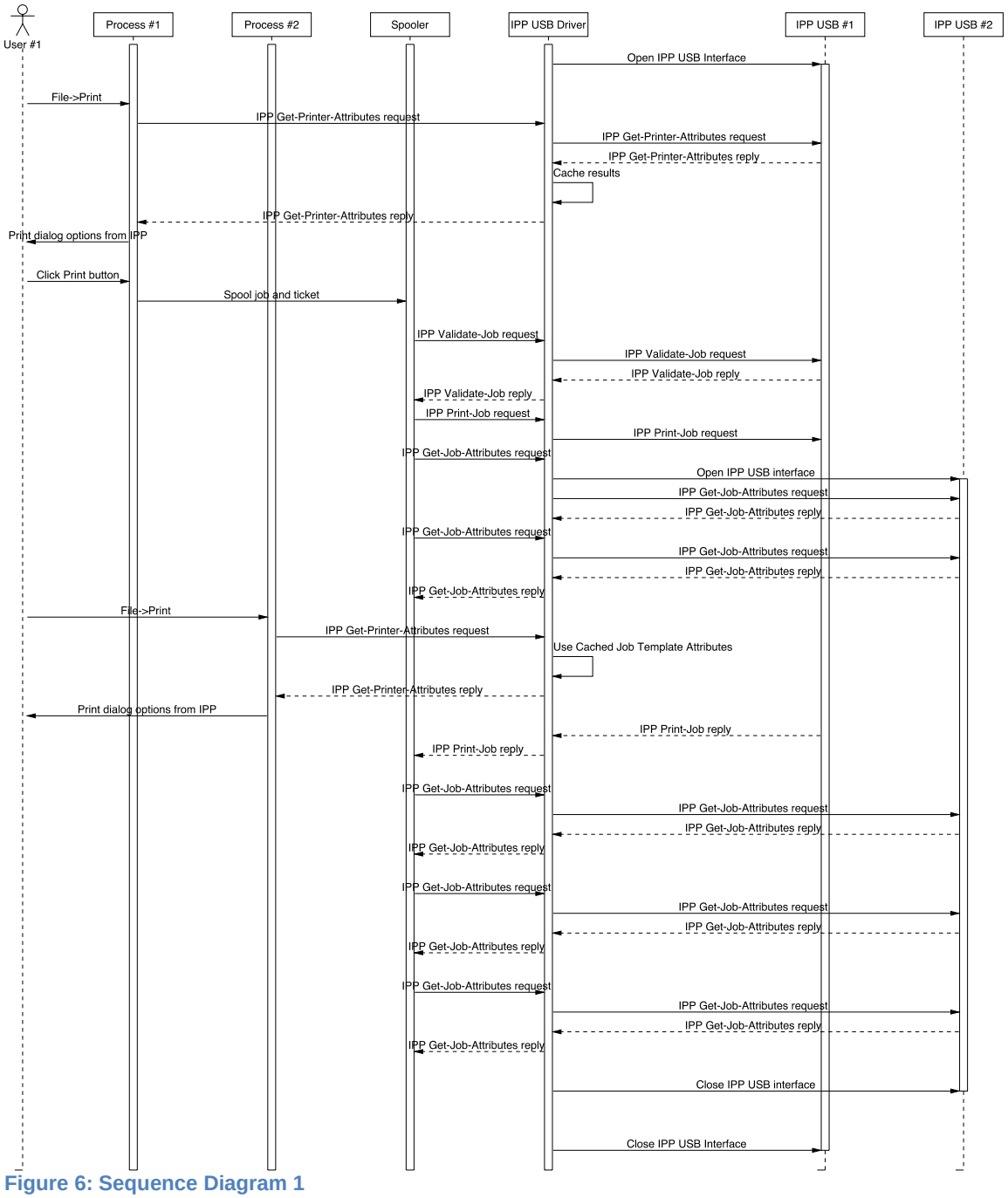

Figure 6: Sequence Diagram 1

C.3 USB Host and Device Interaction Sequence Diagram 2

This presents a more complex scenario with 2 processes engaged in overlapping print dialog presentation and job submission to the print spooler.

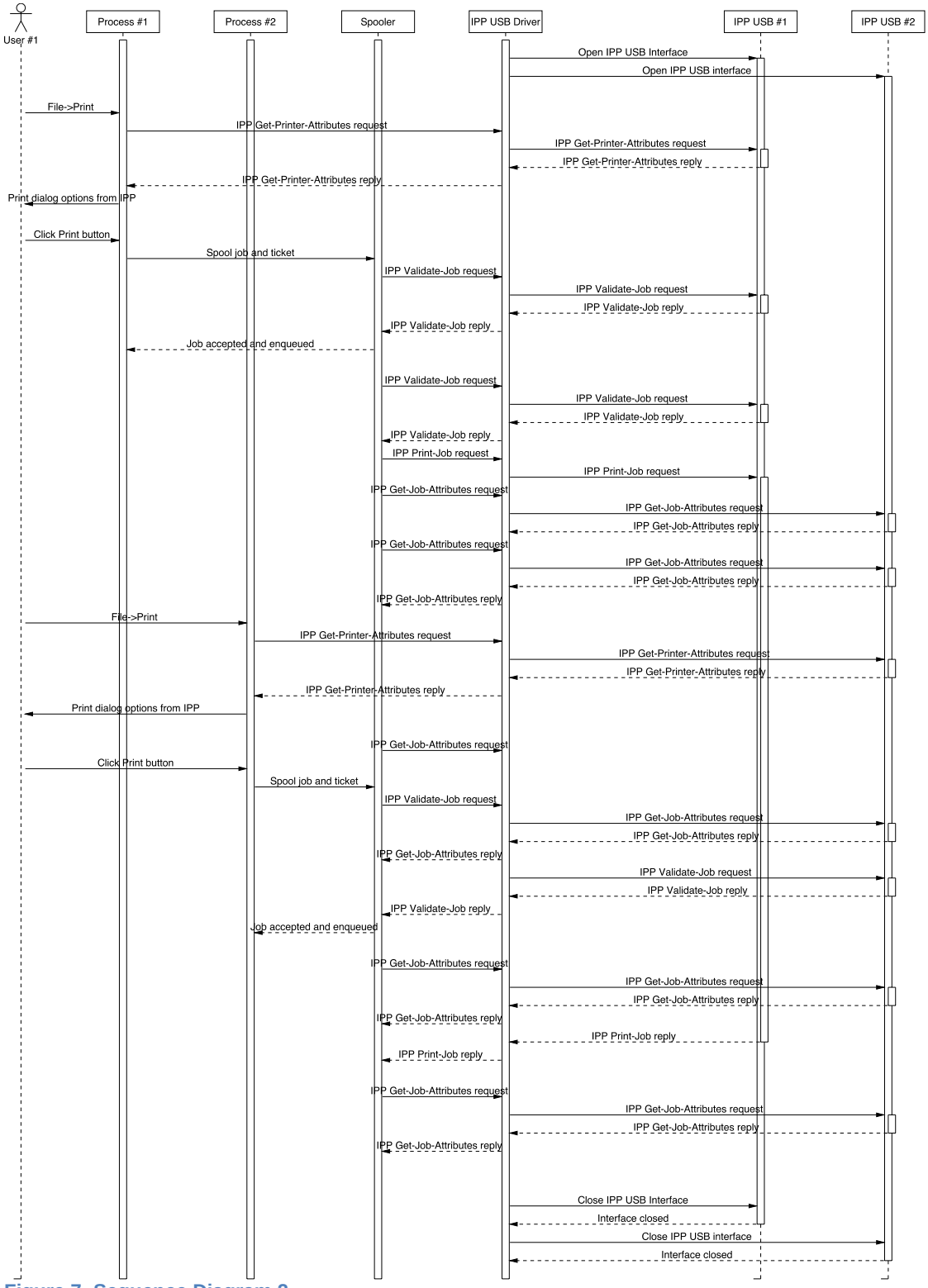

Figure 7: Sequence Diagram 2