IJS Protocol Specification

Version 0.32 — 17 Jan 2002

Raph Levien

This document contains a specification for the IJS protocol, which is intended to make it easier to deploy raster-based printer drivers in a wide variety of environments, including Unix desktops.

1. Introduction

IJS is, first and foremost, a protocol for transmission of raster page images. The protocol is a fairly traditional client-server design. In general, the client sends one or more page images to the server, along with various metadata. Communication is through simple “commands”, which are essentially size-prefixed packets. The client sends a command to the server, then waits for a response command, either ACK or NAK.

Since, in the typical IJS scenario, there is only one client for any given server, it may be helpful to denote the client role as "master" and the server role as "slave". However, this document uses the terms "client" and "server".

On Unix systems, the server “speaks” IJS through stdin and stdout. One consequence of this design decision is that the server can be invoked remotely, for example through **ssh**.

* *It’s not clear yet how useful this will be, but at least people can experiment with it.*

Other forms of communication (such as domain sockets) may be useful, as well, but are not specified in this version.

There are also a large number of things that the IJS specification does *not* address. It does not provide strings for a UI (although the parameter names and values may be used as a fallback when higher-level mechanisms designed to provide these fail). It does not address the task of discovering printers or drivers. It is not designed to dispatch jobs to multiple printers. It does not provide queue management features. It does not address higher level imaging models, or fonts. These are important components of a printing system, and should be addressed by other modules and interfaces.

2. Wire protocol

After a brief initialization handshake, all IJS communication occurs through *commands*. Most of these are sent from the client to the server, but three (IJS_CMD_PONG, IJS_CMD_ACK, and IJS_CMD_NAK) are sent from the server to the client.

With the exception of IJS_CMD_PING, the appropriate response to a command sent from the server is either IJS_CMD_ACK or IJS_CMD_NAK.

The initialization handshake is as follows. First, the client sends the string "IJS\n\252v1\n" (with C backslash escaping). Upon receipt of this string, the server sends the string "IJS\n\253v1\n". At this point, the client may send IJS commands to the server.

IJS is designed to have a simple wire encoding. Integers are encoded as 32-bit big-endian (ie “network order”) values. The encoding for a command is as follows:

Table 1. Wire Encoding of IJS Commands

Command	4-byte integer
Size in bytes	4-byte integer
Arguments ...	

The arguments are simply concatenated. For variable size arguments, the size is either explicitly given as another argument, or, in the case of the last argument, is inferred from the size of the command.

A wire encoding for a typical command is given below. This command sets the Dpi parameter to 600.

Table 2. Example Wire Encoding

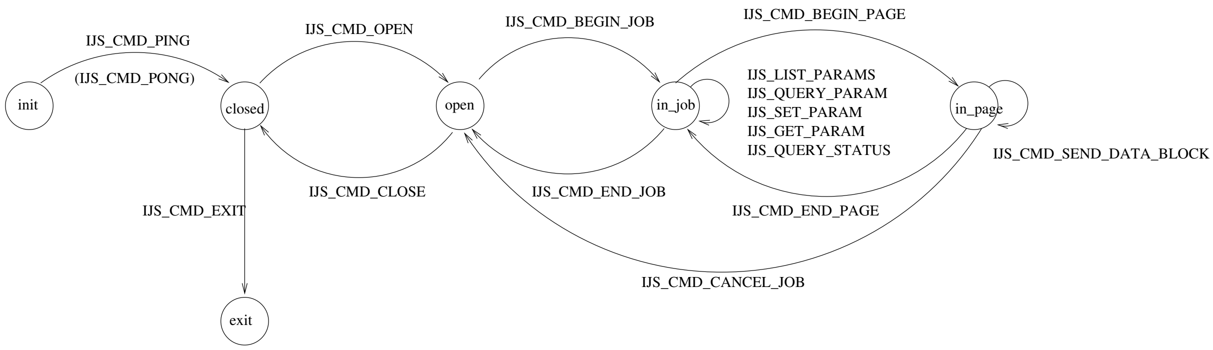
Encoded bytes	Field	Value
00 00 00 0c	Command	IJS_COMMAND_SET_PARAM
00 00 00 16	Size in bytes	22
00 00 00 00	Job id	0
00 00 00 03	Size of parameter name	3
44 70 69	Parameter name	Dpi
36 30 30	Value	600

The numerical values of the commands are:

Table 3. Numerical Values of IJS Commands

Command	Value
IJS_CMD_ACK	0
IJS_CMD_NAK	1
IJS_CMD_PING	2
IJS_CMD_PONG	3
IJS_CMD_OPEN	4
IJS_CMD_CLOSE	5
IJS_CMD_BEGIN_JOB	6
IJS_CMD_END_JOB	7
IJS_CMD_CANCEL_JOB	8
IJS_CMD_QUERY_STATUS	9
IJS_CMD_LIST_PARAMS	10
IJS_CMD_ENUM_PARAM	11
IJS_CMD_SET_PARAM	12
IJS_CMD_GET_PARAM	13
IJS_CMD_BEGIN_PAGE	14
IJS_CMD_SEND_DATA_BLOCK	15
IJS_CMD_END_PAGE	16
IJS_CMD_EXIT	17

A state transition diagram for servers supporting a maximum of one active job at a time is given below:



2.1. IJS_CMD_ACK

This command is sent from server to the client in response to a command from the client, to indicate successful completion. There are no arguments specific to this command. However, for commands (such as IJS_CMD_GET_PARAM) which request a value, this value is returned as the argument in an ACK command.

2.2. IJS_CMD_NAK

This command is sent from server to the client in response to a command from the client, to indicate an error. There is one integer argument, which is the error code. A list of error codes is given in Section 6.

2.3. IJS_CMD_PING

The PING command is sent from the client to the server as part of the connection setup. It contains one integer argument, which is the 100 times the real-valued version number of the largest IJS protocol understood by the client. Thus, for the version of the protocol described in this document, the argument is 30. The appropriate response to a PING is a PONG.

2.4. IJS_CMD_PONG

The PONG command is sent from the server to the client in response to the PING command. It contains one integer argument, which is 100 times the largest IJS version number understood by the server. After a PING/PONG handshake, both client and server should use the minimum of the two version numbers. This negotiation mechanism preserves the ability to make deep changes in future version of the protocol, while preserving backwards compatibility for both clients and servers.

2.5. IJS_CMD_OPEN

The client should send an OPEN command to the server to indicate that printing is imminent. The server can allocate resources, such as tables, at this time.

2.6. IJS_CMD_CLOSE

The client should send a CLOSE command to the server to indicate that printing is complete for now. The server can free any allocated resources at this time.

There should not be any jobs open at the time of the CLOSE command. Handling of any such jobs is undefined.

2.7. IJS_CMD_BEGIN_JOB

The client should send a BEGIN_JOB to the server to begin a job. There is one integer argument, a job id. This id is allocated by the client, and jobs are uniquely identified by the (client, job id) tuple. This job id is present as an argument for all the commands which operate within the context of a job. This job id is valid until the corresponding END_JOB command is invoked, at which point it can be reused if desired.

The connection must be in an open state to begin a job, ie an OPEN command must have been sent, without a corresponding CLOSE.

Servers can choose whether or not to implement multiple jobs, depending on their sophistication. When the number of jobs supported is exceeded, the server should return an IJS_ETOOMANYJOBS error code.

2.8. IJS_CMD_END_JOB

The client should send an END_JOB command to the server on the completion of a job. The one argument is the job id. This command cannot be used in the middle of a page, i.e. when a BEGIN_PAGE command has been issued without its corresponding END_PAGE.

2.9. IJS_CMD_CANCEL_JOB

This command cancels a job. The one argument is the job id. This command can be used whether or not a page is open.

2.10. IJS_CMD_QUERY_STATUS

This command queries the status of a job, or general printer status within a job context. The one argument is the job id. The return data is the printer status.

The format of the printer status is yet to be determined. Glen Petrie has made a proposal on the inkjet-list mailing list. Michael Sweet has suggested adopting the IPP status codes. That standard is fairly rich in status queries. There appear to be at least four queries related to this IJS command: printer-state (enum), printer-state-reasons (keyword), printer-state-message (text), printer-is-accepting-jobs (boolean).

2.11. IJS_CMD_LIST_PARAMS

This command queries the server for a complete list of parameters. Note that this list may change dynamically, in response to setting various parameters, or external events. The argument is the job id. The return value is a comma-separated list of parameters.

2.12. IJS_CMD_ENUM_PARAM

This command queries the possible values for a given parameter. The arguments are the job id and the name of the parameter. The return value is a comma-separated list of values, with the default given first.

Some parameters may not have a small finite enumeration. In these cases, the server should report IJS_ERANGE.

Note also that the comma-separated encoding does not provide a way to report values containing commas. Thus, these should be avoided.

2.13. IJS_CMD_SET_PARAM

This command sets a parameter. There are four arguments: the job id, the size of the parameter name (in bytes), the parameter name, and the value. The size of the value is inferred from the size of the command.

If the parameter is unknown, the server returns an IJS_EUNKPARAM error. If the parameter is known but the value is not appropriate, the server returns an IJS_ERANGE error.

2.14. IJS_CMD_GET_PARAM

This command retrieves the current value of a parameter. There are two arguments: the job id and the parameter name. The value of the parameter is returned.

If the parameter is unknown, the server returns an IJS_EUNKPARAM error.

2.15. IJS_CMD_BEGIN_PAGE

This command begins a new page. All of the parameters affecting the data format of the page should have already been set by this time.

2.16. IJS_CMD_SEND_DATA_BLOCK

This command sends a block of data, in the format defined by PageImageLanguage and its subsidiary parameters. There are no alignment restrictions. There are two arguments: the job id, and the size of the data block in bytes. The data block follows the command, in the same stream.

Note that shared-memory transport of bulk data is anticipated in a future version of this standard. Pipe transport will still be used as a fallback in case shared-memory transport is unavailable.

The server must be in the middle of a page (ie BEGIN_PAGE without the corresponding END_PAGE) when this command is issued.

2.17. IJS_CMD_END_PAGE

This command ends a page. The server must be in the middle of a page when this command is issued. The argument is the job id.

2.18. IJS_CMD_EXIT

This command signals the end of the IJS connection. In the typical case of a server with a single client, the server process terminates upon receipt of this command.

The connection must be in a closed state at the time of this command.

* *Need to look into race condition.*

3. Parameters

IJS defines a small set of standard parameters, which all clients and servers are expected to understand. Individual implementations may extend this standard set with additional parameters specific to the device or driver. Clients should, in general, provide some mechanism for setting (and possibly querying) arbitrary additional parameters. In particular, command line clients should accept command line options to set additional parameters. Interactive clients should ideally query the server for a list of these parameters to display in the interface, then query each parameter for the list of possible values, presented as menu choices.

In addition, in many scenarios, the client may have additional information specific to the device, obtained through other means, for example a PPD (or PPD-like) file specified by the user. Such file formats are well beyond the scope of this specification. However, many users may find the simple parameter mechanism of IJS to be sufficient for their needs. A particular strength of the IJS parameter mechanism is that no additional effort is required to handle dynamic capability information, for example the presence of a hot-pluggable duplexer.

Often, one parameter will be subsidiary to another. In this case, the subsidiary parameter should be set, gotten, or enumerated after the other parameter is set.

4. Standard parameters

This section describes the standard parameters specified by IJS.

4.1. OutputFile

This parameter is the filename intended for IJS output. It will often refer to a device, but can also be a regular file.

Note that this parameter should be considered security-sensitive. Clients should take care to ensure that it is set only to legitimate values.

4.2. OutputFD

This is an alternative to OutputFile, and is intended to support -sOutputFile=- and -sOutputFile="|cmd" configurations of Ghostscript. The parameter is a numeric file descriptor.

4.3. DeviceManufacturer

This parameter is the manufacturer of the printer. In general, it should match the "MANUFACTURER" (or "MFR") field of the IEEE 1284 Device ID string exactly[IEEE1284].

There are many different scenarios for setting and querying this parameter, depending on what information is known about the device.

In the case where the server is able to identify the device, for example by retrieving the IEEE 1284 Device ID string, or through the GET_DEVICE_ID request of the USB Printer Class[USBPrint], getting the value of the parameter will retrieve this identification string. In general, the server should perform the device ID query at the time of the GET_PARAM command.

In the case where the device identification is configured by the client, the client may set this parameter, then set the DeviceModel parameter.

Finally, enumerating this parameter returns a list of manufacturers known by the server. This may be helpful for installing a new printer in cases where automatic device identification is not available.

There may be cases where the server is able to automatically identify the device, and the client attempts to override this identification. The server should allow this override to occur, particularly when the device ID is not one known to the server. However, the server can reject such attempts by returning an IJS_ERANGE error.

4.4. DeviceModel

This parameter is the model name of the printer, and together with DeviceManufacturer, identifies the device. In general it should match the "MODEL" (or "MDL") field of the IEEE 1284 Device ID string.

Usage scenarios are similar to DeviceManufacturer. This parameter is subsidiary to DeviceManufacturer.

Setting the device manufacturer and model may have profound effects on the list of other parameters available. For example, the server may in fact be a wrapper that invokes the “real” server once the device id is known, and then proxies all IJS commands subsequently. Thus, all other parameters other than OutputFD, OutputFile, and DeviceManufacturer, should be considered subsidiary to this one.

4.5. PageImageFormat

This parameter specifies the format of the page image data to be sent to the printer. This standard only defines one standard value: "Raster". Other values, including compressed raster formats, as well as possibly higher level page description languages such as PostScript and PDF, are envisioned as possible future extensions.

When it makes sense, names consistent with the "COMMAND SET" (or "CMD") field of the IEEE 1284 Device ID string are recommended. However, this namespace has many shortcomings for use with IJS. In particular, it tends to identify the command set too vaguely. For example, many Epson printers report merely "ESCPL2", which is not nearly precise enough to usefully drive the printer.

When the value is "Raster", the following parameters are required, and are subsidiary to this one: Dpi, Width, Height, BitsPerSample, ColorSpace, and NumChan.

4.6. Dpi

This parameter is the resolution for transfer of raster data. It is specified as a horizontal resolution, in floating decimal dpi units, an "x", and a vertical resolution, in floating decimal dpi units. Thus, a typical value is "1440x720".

Note that the server may perform scaling of the raster data as part of its processing, before sending it to the device. In these cases, the Dpi parameter specifies the resolution prior to scaling. For example, a driver might accept 720 dpi raster data, then perform 2:1 horizontal pixel replication to drive the actual device at 1440x720 dpi. In this example, the value of the Dpi parameter is "720x720".

4.7. Width

This parameter is the decimal encoded width of the raster image, in pixels. It MUST be set when PageImageFormat is Raster.

4.8. Height

This parameter is the decimal encoded height of the raster image, in pixels. It MUST be set for raster images.

4.9. BitsPerSample

This parameter is the decimal encoded bit depth of samples for pixel values. It MUST be set for raster images. Valid values include 1-7 (implying client-side dithering of image pixels), 8, and 16 (both implying server-side dithering if needed by the device). In general, the total number of bits per pixel is equal to BitsPerSample times NumChan.

In many cases, querying this parameter will be useful. A “dumb” server may choose not to implement color transform and dithering, leaving these to the client. In this case, the result of the query operation will be a list of bit depths actually supported by the device. Simple devices may report "1", while devices capable of both bilevel and 4-level variable dots may report "1,2".

Note that not all combinations of BitsPerSample and ColorSpace are valid. In particular, BitsPerSample less than 8 in combination with a ColorSpace of sRGB or any other colorimetric color space are not valid. Also for scRGB (also known as sRGB64), 16 is the only valid value.

When the value is 16, the ByteSex parameter is required, and is subsidiary to this one.

4.10. ByteSex

When BitsPerSample is equal to 16, this parameter specifies the byte sex of the raster data. Possible values are "big-endian" and "little-endian".

Enumerating this parameter should list the preferred byte sex as the default (ie first in the comma-separated list). In most cases, this will be the byte sex of the server’s host architecture.

Servers limited to 8 bits of depth need not implement this parameter at all.

4.11. ColorSpace

This parameter is a string identifying the color space of the raster image data. It MUST be set for raster images. Standard values include "DeviceGray", "DeviceRGB", "DeviceCMYK", and "sRGB". Servers should support at least one of these color spaces. Clients should be able to produce raster output if at least one of these color spaces is supported by the server.

* *I think we should have a wide-gamut colorimetric color space in the standard list as well. I like La*b* with a recommended bit depth of 16. Any objections?*

A device may choose to provide more color spaces. For example, 6 color inkjets may provide a "DeviceCcMmYK" space. In general, for a client to use any of these nonstandard spaces requires detailed knowledge of the color rendering characteristics of the device.

Servers should not provide additional color spaces which are merely transforms of the standard color spaces. Examples of such discouraged color spaces are HSV, XYZ, Luv, Yuv, YCC, and colorimetric RGB spaces other than sRGB (TODO: unless we decide to accept scRGB/sRGB64).

4.12. NumChan

This parameter is the number of channels in the chosen color space. In general, it can be determined from the ColorSpace. In particular, DeviceGray implies 1, DeviceRGB and sRGB imply 3, and DeviceCMYK implies 4. Attempting to set a NumChan inconsistent with ColorSpace should result in an error.

4.13. PaperSize

This parameter is in W.WWxH.HH format, in inches, i.e. a string that may be produced by sprintf (str, "%fx%f", width, height). If the server knows the paper size (which is unlikely for inkjets), then getting the parameter will give a good value. In the more common case, get simply returns an error code (todo: probably need to allocate a new one for this). Enumerating this parameter may give a list of paper sizes known by the driver that are plausible for the device.

The result of getting or enumerating PaperSize may change dynamically depending on the DeviceModel, Duplex, and possibly “extension” parameters such as those for selecting trays.

Note that this parameter is essentially the same as the PageSize page device parameter. The main difference is units (PostScript uses 1/72" inch units), and the minor syntax nit of PostScript array encoding.

4.14. PrintableArea

This parameter is in W.WWxH.HH format, and describes the printable area of the page. It is expected that the client will usually get it from the server. Any attempt to set it is allowed to fail with an error, even if it's the same value as the get. The value may change dynamically depending on PaperSize and other parameters.

4.15. PrintableTopLeft

This parameter is in W.WWxH.HH format, and contains the left and top margins of the printable area with respect to the media. It is the companion to PrintableArea (I'm considering having a single parameter that ASCII encodes the four floats).

4.16. TopLeft

This parameter, in W.WWxH.HH is intended to be set, and controls the placement of the raster image on the page. The corresponding size of the raster image area can be inferred from the Width, Height, and Dpi parameters.

4.17. PostScript Page Device Parameters

PostScript defines a number of page device parameters, many of which are relevant to IJS, whether using PostScript or not. Further, many proposals for characterizing device capabilities are based on PPD files, which use a consistent namespace and semantics to page device parameters.

IJS imports the namespace of PostScript page device parameters, prefixing it with the string "PS:". The client can assume that any parameters returned by a LIST_PARAMS command matching this prefix are in fact PostScript page device parameters. Values are straightforward ASCII encodings. For example, arrays are encoded as space-separated values, enclosed in square brackets. The set of valid page device parameters is defined in the PostScript Language Reference Manual[PLRM], particularly Chapter 6.

Some page device parameters are subsumed by native IJS parameters, and should not be used. These include PageSize (subsumed by PaperSize), ProcessColorModel (subsumed by ColorSpace), Margins and PageOffset (subsumed by TopLeft), and HWResolution (subsumed by Dpi).

Devices supporting duplexing should implement PS:Duplex and PS:Tumble, both booleans. A value of true for PS:Duplex requests printing on both sides of the page. When PS:Duplex is true, PS:Tumble specifies the relative orientation of the pages. When PS:Tumble is false, the pages are oriented suitably at the left or right. When PS:Tumble is true, the pages are oriented suitably for binding at the top or bottom. Enumerating the PS:Duplex parameter should return a single "false" value when the server knows that the device is not capable of duplexing, and either "false,true" or "true,false" if it may be.

* *Note that the HPIJS 1.0 implementation of IJS, identifying itself as IJS version 0.29, specifies an integer-valued Duplex parameter, with values of 0 (PS:Duplex = false, PS:Tumble don't care), 1 (PS:Duplex = true, PS:Tumble = false), and 2 (PS:Duplex = true, PS:Tumble = true). An integer valued Duplex parameter is inconsistent with the PostScript specification. However, clients desiring compatibility should set the integer-valued Duplex parameter rather than the PS: parameters when the server reports a version of 0.29.*

Devices supporting roll-fed media should implement PS:RollFedMedia, PS:Orientation, PS:AdvanceMedia, PS:AdvanceDistance (note that units are integer 1/72"), and PS:CutMedia.

Other parameters that may be useful for some devices include PS:MediaColor, PS:MediaWeight, PS:MediaType, PS:MediaClass, PS:InsertSheet, PS:LeadingEdge, PS:ManualFeed, PS:TraySwitch, PS:MediaPosition, PS:ImageShift, PS:MirrorPrint, PS:NegativePrint, PS:NumCopies, PS:Collate, PS:Jog, PS:OutputFaceUp, PS:Separations, and PS:SeparationColorNames. Other parameters are allowed, but are unlikely to be useful in an IJS context.

5. Parameter Namespace Extension

While this document specifies enough parameters to be able to print usefully, there is a huge diversity of devices and applications, often indicating additional parameters not specified. IJS is designed to accomodate these additional parameters as extensions. It is expected that the namespace of these extensions will be managed informally. Note that collisions in this namespace are not necessarily fatal, as many will be device or manufacturer specific, so that the device id may be used to disentangle them. Even so, it is clearly a good idea to manage this namespace well. This section recommends some practices towards this goal.

When possible, extension parameters should be prefixed, with a colon separating the prefix from the base parameter name. Well known prefixes give clients useful information about parameters, even when the client lacks information about the specific parameter. An unknown prefix at least allows the client to identify the parameter as a nonstandard extension.

This document specifies a number of standard prefixes. We also reserve the following prefixes for possible use in future revisions of this protocol: IPP, UPDF. Further, the Omni: prefix is reserved for the Omni group at IBM, and CUPS: is reserved for the CUPS project.

* *Robert, do you want STP:? Anyone else?*

5.1. Quality:

Inkjet printers often provide a rich set of options for tuning output quality, or selecting a point along a speed/quality tradeoff. The details of these options vary widely from device to device. When made available through IJS, they should be grouped under the Quality: prefix.

For example, HPIJS 1.0 has the following parameters, for HP inkjet printers: Quality, MediaType, ColorMode, and PenSet. To be compliant with versions 0.30 and later of IJS, they should be named Quality:Quality, Quality:MediaType, Quality:ColorMode, and Quality:PenSet.

Note that Quality:MediaType overlaps somewhat with PS:MediaType. In general, the former specifies a color profile or printing mode (for example, to optimize printing on transparencies). The latter is often used for selecting a paper source, for example letterhead or envelopes. The former is more likely to be useful in inkjet applications.

The Dpi and ColorSpace parameters are subsidiary to any Quality: parameters provided.

5.2. Finishing:

Finishing options, such as stapling and collating, should be grouped under the Finishing prefix.

The PS page device parameter namespace includes some finishing options, including Duplex, Tumble, Collate, Jog, and the roll-fed parameters: RollFedMedia, Orientation, AdvanceMedia, AdvanceDistance, and CutMedia. For these parameters, the PS: prefix is preferred.

The PPD specification describes a number of additional finishing parameters (section 5.18 of [PPD]). Where possible, Finishing: parameters should be consistent with the PPD specification.

5.3. PPD:

The PPD specification[PPD] contains a large number of options and parameters that may be provided by printers. The PPD: prefix is reserved for PPD parameters that are made available through the IJS protocol.

In cases where both a page device parameter and a PPD parameter specify the same setting, the PS: page device parameter takes priority. In many cases, page device parameters are advantageous because they are designed for both getting and setting, while PPD itself is a static file format. In addition, finishing parameters should be under the Finishing: namespace.

In general, use of the PPD: extension is not recommended, as the PPD file format tends to be specific to PostScript printers.

* *We could use more specific advice on when to use PPD: parameters, and when not to. Anyone with more PPD knowledge willing to help with this?*

6. Error codes

Any IJS command may either succeed or fail. Success is indicated by an IJS_ACK response. Failure is indicated by an IJS_NAK response, which includes an integer error code.

The current draft contains the following error codes:

Table 4. Draft IJS Error Codes

Symbolic definition	Numeric value	Meaning
IJS_EIO	-2	I/O error
IJS_EPROTO	-3	protocol error
IJS_ERANGE	-4	out of range
IJS_EINTERNAL	-5	internal error
IJS_ENYI	-6	not yet implemented
IJS_ESYNTAX	-7	syntax error
IJS_ECOLORSPACE	-8	unknown color space
IJS_EUNKPARAM	-9	unknown parameter
IJS_EJOBID	-10	job id doesn't match
IJS_ETOOMANYJOBS	-11	reached limit of server's #jobs
IJS_EBUF	-12	buffer isn't big enough

However, I see that this list overlaps the status codes for IPP operations (section 13.2 of [RFC 2911]) to a large extent. I am strongly considering unifying these.

7. Acknowledgements

IJS is directly inspired by the HPIJS work done by the HP Vancouver team, particularly David Suffield. This spec also benefited from comments and suggestions from Robert Krawitz, Grant Taylor, Glen Petrie, Russell Lang, Michael Sweet, and the Omni team at IBM: Mark VanderWiele, Mark Hamzy, and Pete Zannucci.

* *Please add your name here. Incidentally, the <ackno> tag of DocBook seems more reasonable than a section, but I can't get it to format with a nice title.*

References

[RFC 2911] T. Hastings, R. Herriot, R. deBry, S. Isaacson, and P. Powell, *Internet Printing Protocol/1.1: Model and Semantics*, September 2000.

[IEEE1284] *IEEE Std.1284-1994 Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers*, 1994.

[USBPrint] *Universal Serial Bus Device Class Definition for Printing Devices*, 1.1, January 2000.

[PLRM] *PostScript Language Reference*, third edition, Adobe Systems Incorporated, Addison-Wesley, 1999.

[PPD] *PostScript Printer Description File Format*, version 4.3, Adobe Systems Incorporated, Technical Note 5003, 9 February 1996.