Why use HTTP?

- Builds on existing, proven technology
- Immediately works with existing components
  - Web Browsers
  - Web Servers
  - Proxies, Gateways, Firewalls

- very rapid prototyping/development possible using
  - HTML Forms
  - CGI and/or existing Server APIs

- Leverages other work done for the web
  - security (SSL, SET, SHTTP)
  - directory (LDAP)
  - naming (URLs)

Concerns with HTTP that have been raised include

- HTTP is stateless
- Performance/network traffic

But ...

- Maintaining state is not required across connections for Print operations
- Performance problems are largely caused by the multiple connections created by Browsers as they render a page of HTML. There is nothing intrinsic in the protocol that makes performance a problem.

Web Server Interfaces

CGI

When the POST method is used the data is actually carried in the body part of the HTTP message. The server passes the length of the data and the content type to the CGI application in the environment variables "CONTENT_LENGTH and "CONTENT_TYPE", respectively. The data itself is passed to the CGI application's *stdin.*

When the GET method is used, the data is transported as part of the URL, and is also passed to the CGI application in an environment variable, QUERY_STRING.

In either case, the data to be passed to the CGI application is normally encoded as

name=value&name=value&name=value& ...

However, note that a CGI application ***does not require*** the data to be coded this way.

The data stream follows URL encoding rules, that is, spaces are replaced by plus signs and certain special characters are converted to hexadecimal.

The CGI application used to process the data must be named in the URL. For example, sending a request to some.domain.com and requesting that the application "IPP.exe" be used to process the content of the HTTP message would require the URL

"http://some.domain.com/ipp.exe"

Server APIs (e.g. Microsoft ISAPI or Netscape NSAPI)

Microsoft, Netscape and presumably other Web server developers, have provided APIs in their servers which give programmable access to HTTP data without using the CGI interface. This has been done specifically to allow for easy extendability of the server capabilities and to remove performance issues related to the use of the CGI interface. These APIs pass information, such as content type and length, in data structures defined by the API. Operations are then provided in the API to *read* the data in the message body.

Microsoft requires that such applications be written as DLLs. In this case, the URL specifies the DLL as

"http://some.domain.com/ipp.dll"

Scoping Philosophy

One could approach the scoping rules for IPP datastreams in a couple of different ways. The choice of scoping philosophy will impact the encoding of the datastream. I think of 2 simple cases:

Case I: Scoping, if provided, is implicit and defined by position in the datastream.

- *Header stuff*
- *Job attributes*
- *Document 1 attributes*
- *Document 1*
- *Document 2 attributes*
- *Document 2*

Case 2: Scoping is explicit and defined by markers and lengths in the data stream.

- *Header stuff*
- *Begin Job*
  - *Job attributes*
  - *Begin Document 1*
    - *Document 1 attributes*
    - *Document 1 content*
  - *Begin Document 2*

Obviously all kinds of combinations and encodings are possible. Although case 1 appears simpler, I believe that case 2 is the better approach. It makes extension easier, provides for validity checking of the content and strict application of scoping rules, and I think is easier to parse.

HTML/Forms vs. Imbedded Protocol

Using helper applications or Java on the client , combined with the use of Server applications written to the server APIs provide a tremendous amount of flexibility without requiring changes to existing browsers or servers. Assuming that both my client side helper application and my server side application conform to a standard concrete syntax, it shouldn't matter what the syntax looks like, just that both use the same one. Thus, I could use either the standard CGI syntax as described previously, or the imbedded protocol defined in the current IPP draft.

If I insist on using a standard browser, without a helper application, and depend on html forms as the client side datastream generator, then I'm pretty well stuck with standard CGI syntax. Using the CGI syntax, the Print Request example in the IPP appendix would look something like the following, the stuff in the box being the IPP unique stuff.

```
Post http://some.domain.com/ipp.exe/printer-1  http/1.0
 Content-type = Application/IPP
```

```
  "request-line"=Print+IPP/1.0
  "job_name"=My+Job
  "medium"=iso-a4-white
  "notification-events"=Job-completion
  "notification-address"=Joe@pc.domain.com
  "document_name"=letter+to+Mom
  "Document_Format"=Postscript/2.0
  "Document_content=%!PS-Adobe-3.0
%%Title: PowerPoint -Ipp2.ppt
%%Creator: PSCRIPT.DRV Version 4.0
%%CreationDate: 12/04/96 12:24:21
%%BoundingBox: 19 9 593 784
%%Pages: (atend)
%%PageOrder: Special
%%Requirements:
%%DocumentNeededFonts: (atend)
%%DocumentSuppliedFonts: (atend)
%%DocumentData: Clean7Bit
%%LanguageLevel: 1
%%EndComments
...
```

URL Encoding Problem

Given the previous flow, I immediately note that there is a problem with the document content being passed in standard CGI format. To do so would require that we force URL encoding rules on the document content. Otherwise it is not valid CGI input. This would require that all blanks in the sample Postscript file be replaced with plus signs and certain special characters would be converted to hexadecimal. This would be pretty disastrous! It is also not clear what will happen when the parser finds CRs and LFs in the PostScript file.

Post http://some.domain.com/ipp.exe/printer-1 http/1.0
Content-type = Application/IPP

```
"request-line"=Print+IPP/1.0
"job_name"=My+Job
"medium"=iso-a4-white
"notification-events"=Job-completion
"notification-address"=Joe@pc.domain.com
"document_name"=letter+to+Mom
"Document_Format"=Postscript/2.0
"Document_content=%!PS-Adobe-3.0
%%Title:+PowerPoint +-Ipp2.ppt
%%Creator:+PSCRIPT.DRV+Version 4.0
%%CreationDate:+12/04/96+12:24:21
%%BoundingBox:+19+9+593+784
%%Pages:+(atend)
%%PageOrder:+Special
%%Requirements:
%%DocumentNeededFonts:+(atend)
%%DocumentSuppliedFonts:+(atend)
%%DocumentData:+Clean7Bit
%%LanguageLevel:+1
%%EndComments
...
```

Postscript file gets clobbered by superimposing URL encoding rules (and maybe by getting rid of CRs and LFs.

Aha, but why not carry the document as a separate mime encoded message?

I think this is the suggestion that Bob Herriot was making. My apologies Bob if I have not understood correctly. However, one could send the IPP content as a mutipart/mixed encoded message. The job specification could be sent as text and the postscript job as -- well, postscript. This would appear as

```
Post http://some.domain.com/ipp.exe/printer-1  http/1.0
 Content-type = Multipart/mixed;
   boundary=unique-boundary-1
```

```
content-type=text/plain; charset=US-ASCII
"request-line"=Print+IPP/1.0
  "job_name"=My+Job
  "medium"=iso-a4-white
  "notification-events"=Job-completion
  "notification-address"=Joe@pc.domain.com
  "document_name"=letter+to+Mom
--unique-boundary-1
```

```
content-type=application/postscript;
  boundary=unique-boundary-2
%!PS-Adobe-3%%Title: PowerPoint -Ipp2.ppt
%%Creator: PSCRIPT.DRV Version 4.0
%%CreationDate: 12/04/96 12:24:21
%%BoundingBox: 19 9 593 784
%%Pages: (atend)
%%PageOrder: Special
%%Requirements:
%%DocumentNeededFonts: (atend)
%%DocumentSuppliedFonts: (atend)
%%DocumentData: Clean7Bit
%%LanguageLevel: 1
%%EndComments
...
--unique-boundary-2
```

The structure Problem

Using Multipart/mixed mime types does seem to help, but I see two additional. potential problems:

Post http://some.domain.com/ipp.exe/printer-1 http/1.0
Content-type = Multipart/mixed;
boundary=unique-boundary-1

```
content-type=text/plain; charset=US-ASCII
"request-line"=Print+IPP/1.0
  "job_name"=My+Job
  "medium"=iso-a4-white
  "notification-events"=Job-completion
  "notification-address"=Joe@pc.domain.com
  "document_name"=letter+to+Mom
--unique-boundary-1
```

The standard does not allow for related body parts. Thus there is no way, except by convention to know that the job part is related to the document part

boundary must be a unique string not found in the data. This requires prescanning the document to guarantee a unique string can be determined.

```
content-type=application/postscript;
  boundary=unique-boundary-2
%!PS-Adobe-3%%Title: PowerPoint -Ipp2.ppt
%%Creator: PSCRIPT.DRV Version 4.0
%%CreationDate: 12/04/96 12:24:21
%%BoundingBox: 19 9 593 784
%%Pages: (atend)
%%PageOrder: Special
%%Requirements:
%%DocumentNeededFonts: (atend)
%%DocumentSuppliedFonts: (atend)
%%DocumentData: Clean7Bit
%%LanguageLevel: 1
%%EndComments
...
--unique-boundary-2
```

These problems could be showstoppers when shipping multiple documents, or when eventually extending the standard to allow mixed job attributes on a document basis.

So ... back to an imbedded protocol

Given the existance of server apis and cgi we can still go back to the imbedded protocol approach, without impacting current server implementations. This does require a helper application or a Java application to generate the appropriate syntax on the client side.

Post http://some.domain.com/ipp.exe/printer-1 http/1.0
Content-type = Application/IPP

Defining a new mime type gives us the freedom to define what our syntax looks like - not be forced to live with what has been defined for another application.

```
Print  IPP/1.0
   Print_Job_Header
      Content_length=56787
      job_name=My Job
      medium=iso-a4-white
      notification-events=Job-completion
      notification-address=Joe@pc.domain.com
      document_name=letter to Mom
    Document_Header
      Document_length=56653
      Document_Type=postscript;
        %!PS-Adobe-3%%Title: PowerPoint -Ipp2.ppt
        %%Creator: PSCRIPT.DRV Version 4.0
        %%CreationDate: 12/04/96 12:24:21
        %%BoundingBox: 19 9 593 784
        %%Pages: (atend)
        %%PageOrder: Special
        %%Requirements:
        %%DocumentNeededFonts: (atend)
        %%DocumentSuppliedFonts: (atend)
        %%DocumentData: Clean7Bit
        %%LanguageLevel: 1
        %%EndComments
```

Length allows me to find the end of the job. In the future I could now include multiple jobs in the same message.

Use of a document header and a document length provide strong scoping rules. Allows future extension to real multiple documents, with different job attributes.

Length allows me to pass the entire document on to another process, without having to scan byte by byte, looking for the end.