Internet Imaging Protocol
Specification, version 1.0, revision 1

1. Overview

The Internet Imaging Protocol (IIP) is designed to send FlashPix™[1] image data and related descriptive information over HTTP and socket-based network connections. This specification is undertaken by Hewlett-Packard Company, Live Picture Incorporated, and Eastman Kodak Company in collaboration with Microsoft, Netscape, and others. This document describes the technical details of the protocol used for requesting and serving image tiles and image information over a network.

The design of the Internet Imaging Protocol was guided by the following principles

- Minimize request/response calls
- Minimize commands and subcommands
- Minimize response types and formats
- Minimize network bandwidth consumed

1.1 Purpose

The Internet Imaging Protocol enables efficient access to multi-resolution images over internets and intranets. Structured to take advantage of the FlashPix image architecture[2], IIP allows a single image file to be used for fast browsing, high-resolution printing, complex image manipulation, and simple snapshot viewing. The IIP functions can access all the image information in the underlying file without requiring extensive server-side processing.

Images stored in FlashPix format are organized into 64 $\times$ 64 pixel 'tiles', stored at multiple resolutions, and located using a 'directory'. This makes it possible to efficiently serve images or sections of images to the client at the resolution best for the particular application. Other information such as content description notes, color space, and other properties are also available via the IIP functions. To limit the number of required requests over a network, certain groups of image properties can be retrieved in one request.

1.2 Specification Organization

This document is divided into several sections, each describing an aspect of the IIP.

- Section 1: Introduction
- Section 2: Request and Response Structure defines the data requests and data responses.
- Section 3: HTTP protocol specification defines the usage of requests and responses over an HTTP connection.
- Section 4: Sockets protocol specification defines the usage of requests and responses over a socket connection.

1.3 General Grammar Rules

- Label names are case-insensitive in both the client request and server response.
- Numeric values are represented as ASCII strings unless otherwise noted as "data-stream". For example, the rational number 1.23 is represented in four bytes as "1.23".
- Dates are represented as strings of the form specified in Internet RFC 1123.

[1] FlashPix is a trademark of the Eastman Kodak Company
[2] See FlashPix Specification 1.0 available at http://www.kodak.com/go/FlashPix

- Requests are concatenated using “&” for HTTP, but are separated by CRLF for sockets connections.
- Socket connections are stateful, while HTTP is stateless between request groups.
- IIP commands are both parsed and executed sequentially (left to right in the case of CGI, and in receipt order for sockets). It is important to note that command modifiers, like those which modify CVT are not executed sequentially; these commands have a pre-defined execution sequence.

2. Command Set

2.1 Syntax

2.1.1 Basic Rules

The following basic rules are used throughout this specification:

- Requests are made using the following syntax defined using modified BNF as in RFC 822 and RFC 1123. It follows the conventions of RFC 822 with the following augmentations:
- `"|"` is used to designate alternatives.
- parenthesis `"("` and `")"` are used to designate associative grouping.
- brackets `"["` and `"]"` are used to designate optional elements.
- The "*" character indicates repetition. The form `"n*m"` indicates repetition of a minimum of n and maximum of m repetitions of the following element. `1*` indicates at least one repetition. `*` indicates zero or more repetitions.
- Non-printable ASCII characters are represented in hex as `\nn`.
- An exact number of repetitions of an element is indicated with a value `<n>` preceding an element, for example `8ALPHA`.
- "<" and ">" are used to indicate descriptive text.
- the symbol "⇒" is used to signify communication from client to server; the symbol "⇐" is used to signify communication from server to client.

```
OCTET           = \00 .. \FF
CHAR            = \00 .. \7F
UPALPHA         = "A".."Z"
LOALPHA         = "a".."z"
ALPHA           = UPALPHA | LOALPHA
DIGIT           = "0" .. "9"
CTL             = \00 .. \1F | \7F
CR              = \0D
LF              = \0A
SP              = \20
HT              = \09
<">             = \22
CRLF            = CR LF
HEX             = DIGIT | "A" .. "F" | "a" .. "f"
```

2.1.2 General Syntax for Requests

Requests for data are made by the client with the following general syntax. Multiple requests can be concatenated as appropriate within the underlying protocol.

```
COMMAND         = 3ALPHA
REQUEST         = COMMAND "=" 1*UCHAR
OLABEL          = NAME *("," RANGE)
NAME            = 3ALPHA *(DIGIT | ALPHA | "-")
PATH            = ["/"] FSEGMENT *("/" SEGMENT)
PARAMNAME       = ALPHA | DIGIT | ESCAPE | SAFE
FSEGMENT        = 1 * PCHAR
SEGMENT         = * PCHAR
```



```
PCHAR           = UCHAR | "@" | "="
UCHAR           = UNRESERVED | ESCAPE
UNRESERVED      = ALPHA | DIGIT | SAFE | EXTRA
ESCAPE          = "%" HEX HEX
SAFE            = "$" | "-" | "_" | "." | "+"
EXTRA           = "!" | "*" | "'" | "|" | "(" | ")" | ","
RANGE           = "*" | INT | CONTRANGE
CONTRANGE       = INT "-" INT
INT             = 1 * 10 DIGIT <A maximum of 2^31-1, or 2,147,483,647>
FLOAT           = ["-"] INT | * DIGIT "." 1 * DIGIT | 1 * DIGIT "." * DIGIT
STREAM          = * OCTET
PID             = 8HEX
```

2.1.3 General Syntax for Responses

Responses to requests for image data are sent from the server to the client. Multiple responses can be concatenated as described in Sections 3 and 4.

In cases where variable length data must be transmitted, the last `INT` in the `LABEL` indicates the data length in `OCTETs`. The terminating `CRLF` pair is not counted. Not every `LABEL` has an `INT`, but the last `INT` appearing in a `LABEL` is always data length.

In cases where the response length is not variable, the response is read as an `OCTET` stream up to, but not including, the first `CRLF`.

```
RESPONSE        = 1 * ENTRY
ENTRY           = LABEL ["/" LENGTH] ":" STREAM CRLF
LABEL           = NAME IDENT
IDENT           = * ("," PARAMNAME)
CLASSID         = "{" 8HEX "-" 4HEX "-" 4HEX "-" 4HEX "-" 12HEX "}"
STRING          = * (UCHAR | CRLF)
DATE            = <as per RFC 1123>
RESERVED        = ";" | "/" | "?" | ":" | "@" | "&" | "="
PARAMNAME       = ALPHA | DIGIT | ESCAPE | SAFE
STREAM          = * OCTET
PID             = 8HEX
ACKNOWLEDGE     = "OK" CRLF <sockets implementation only>
END             = "END" CRLF <sockets implementation only>
```


2.2 Basic Commands

These commands must be supported by all servers.

FIF

Purpose	Specify the name of the image file.
Syntax	`FIF=path`
Input Parameters	`PATH` *`path`* Valid relative path to the image file from the server's root directory. Paths can not include "//" or ".."
Response	`FIF` returns nothing when accompanied by other commands. If `FIF` is the sole key-value pair, the server responds as if it had received an `OBJ=Basic-info` command.
Example	`FIF=/myimage/finepicture.fpx`
Notes	The server's configuration defines an `FPX_ROOT` for images accessed using IIP. This is the path to which requests for `FIF=/` are translated. Since all paths are considered absolute, `FIF=finepicture.fpx` is equivalent to `FIF=/finepicture.fpx` This is often set to the document root. This key-value pair is always present in an HTTP IIP request, unless alternative image path parsing techniques are utilized, such as Netscape Corporation's NSAPI. The path delimeter in IIP is not platform specific – it is always '`/`'

OBJ

Purpose	Request an object from the server.
Syntax	`OBJ=object`
Input Parameters	`OLABEL object` The name of a requested data object from the table in Section 2.6.
Response	varies by object. See Section 2.6.
Example	⇒ `OBJ=Resolution-number` ⇐ `Resolution-number:6`
Notes	The exact syntax and format of an `OBJ` request varies by object.

TIL

Purpose
Request one or more tiles from the server in their native FlashPix format.

Syntax
`TIL=res,tile[,sub]`

Input Parameters
`RANGE` *`res`*
the desired resolution

`RANGE` *`tile`*
the desired tile(s)

`RANGE` *`sub`*
the desired subimage. If omitted, this defaults to the primary subimage.

Response
`Tile,res,sub,tile/length:data`
`INT` *`res`*
resolution number of tile

`INT` *`sub`*
subimage number of tile

`INT` *`tile`*
tile number

`INT` *`length`*
length of data

`STREAM` *`data`*
the data stream is composed by pre-pending the 4 byte compression type and the 4 byte compression subtype (see Section 4.1.2 of the FlashPix Specification version 1.0) with the requested tile stream.

Example
⇒ `TIL=4,0-5`
⇐ `Tile,4,0,0/l:<data>`

Notes
A range of requests is returned with separate `TIL` responses.
A range of subimages or resolutions can be used. The subimages or ranges are returned from the primary object store by default, or from the object store indicated using `SDS`; see FlashPix V1.0 section 1.4.
Ranges are always understood to be contiguous in the rectangular rather than numerical sense. This decreases the number of requests needed to access any rectangular region to one. It requires the server to calculate the necessary tiles from the tile corners.
For example, if the image has tiles:

1	2	3	4
5	6	7	8
9	10	11	12

a request for tiles 2-11 will result in tiles 2, 3, 6, 7, 10, and 11.

JTL

Purpose	Retrieve a tile as a complete JFIF image.
Syntax	`JTL=res,tile[,sub]`
Input Parameters	`INT` *`res`* the desired resolution `INT` *`tile`* the desired tile `INT` *`sub`* the desired subimage. If omitted, this defaults to the primary subimage.
Response	a complete JFIF image of that tile.
Example	⇒ `JTL=4,0` ⇐ `<data>`
Notes	A range of tiles may not be requested. Images returned by this request are preceded by the MIME type image/jpeg, but the color space of the returned tile is not guaranteed to be CCIR-601. The server is not obligated to perform color transformation. This command allows client applications to access DCT transform resources which may only be available upon return of MIME type image/jpeg. The response for this command deviates from the BNF definition for RESPONSE.

2.3 Complex Image Commands

These commands must be supported by all servers.

These commands refer, often in detail, to the structured storage model used by the FlashPix format. Structured storage is detailed in Appendix A of the FlashPix Specification version 1.0. In brief, structured storage implements a compound object storage model. It may be thought of as a 'file system within a file'. A file in structured storage contains two kinds of objects; storages and streams. The storages are analogous to directories which contain files. The streams are akin to files. A storage may contain storages and/or streams. The storages within a FlashPix file may be navigated with the commands in this section, and the streams within the storages accessed.

Property sets are streams that contain tagged data. The tags are the property ID's. A binary specification of property sets is contained in section A.2 of the FlashPix Specification version 1.0.

Streams can have many different kinds of information in them. The Class ID of the stream identifies the type of information it contains.

PTY

Purpose: Retrieve the data for the indicated Property

Syntax: `PTY=class,propertya[-propertyb]`

Input Parameters:

`STRING class`
the name of the property set

`PID propertya`
the property ID of a single property, or the starting property of a sequential range

`PID propertyb`
the property ID of the terminating property of a sequential range

Response:

`Property,name,property/length:data`
`STRING class`
the Class ID of the property

`PID property`
the property ID of the property

`INT length`
length in bytes of data

`STREAM data`
requested property as binary data

Example:

```
⇒ PTY=\05Image Info,22000000-22000004
⇐ Property,\05Image Info,22000000/52:(52 bytes of data)
⇐ Property,\05Image Info,22000001/48:(48 bytes of data)
⇐ Property,\05Image Info,22000002/64:(64 bytes of data)
⇐ Property,\05Image Info,22000003/20:(20 bytes of data)
⇐ Property,\05Image Info,22000004/132:(132 bytes of data)
```

Notes: A range of requests is returned with multiple Property returns.

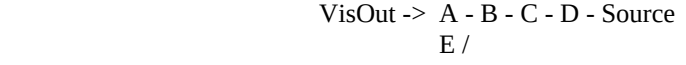
GRP

Purpose: Gets the ‘Rendering Paths’ for the image.

In a FlashPix file, a Rendering path is a set of streams which describe the relationship of an image to its rendering information. A FlashPix image may have multiple rendering paths. GRP returns only the rendering paths referred to in the Visible Outputs property in the Global Information property set.

Each stream in a rendering path may point to one or more other streams. In a network environment, it is undesirable to retrieve a stream, interpret its relationship, and retrieve subsequent streams. This command is intended to allow for the server to traverse the rendering path stream relationship and return all pertinent streams at once.

A FlashPix file contains a transformation graph that specifies how one or more source image objects are transformed and combined to produce one or more result. The graph may have discontinous sections. A FlashPix file also specifies a list of nodes in the graph that are "worthy" of being considered the “results” of the graph (visible outputs). GRP returns all nodes in the graph that are involved in producing the named results. Nodes that are not involved in producing the named results are not returned, even if they are part of a branch of a returned path. For example, the file names node A as a visible output. A is produced by putting image C through transform B. Node D is produced by putting image C through a different transform, E. GRP would return A, B and C only.



E is in the rendering path “referred” to by the visible output (entry for A). However, E should not be returned by GRP.

Syntax `GRP=`*`mask`*

Input Parameters

`INT` *`sub`*
the subimage for which to return the rendering path

`1HEX` *`mask`*
A bitwise mask specifying the type(s) of information to return. The bits in the mask are defined as:

0x01	global information property set
0x02	transform property set(s)
0x04	data object property set(s)
0x08	operation property set(s)

Response `Stream,`*`name/length`*`:`*`data`*

`PARAMNAME name`
name of stream

`INT length`
length of bytes of the data

`STREAM data`
the requested stream

Example

```
⇒ GRP=7
⇐ Stream,\05Global Information/139:(139 bytes of data)
⇐ Stream,\05Data Object 000001/103:(103 bytes of data)
⇐ Stream,\05Data Object 000002/143:(143 bytes of data)
⇐ Stream,\05Data Object 000003/99:(99 bytes of data)
⇐ Stream,\05Data Object 000004/168:(168 bytes of data)
⇐ Stream,\05Transform 000001/145:(145 bytes of data)
⇐ Stream,\05Transform 000002/206:(206 bytes of data)
```

Notes Multiple responses may be returned for a single GRP request. The response is as if one or more STM commands had been made.

SDS

Purpose Sets the data object storage. Subsequent commands will treat the storage pointed to by this command as the root storage.

Syntax `SDS=`*`doid`*`[,`*`doid`*`]`

Input Parameters

`INT` *`doid`*
the data object ID to set as root

Response none for http; an acknowledge for sockets

Example `⇒ SDS=3`

Notes This command can be understood as analogous to the UNIX™[3] `CD` command.
The ‘path’ specified is always defined from the root of the Structured Storage structure (absolute).

[3] UNIX is a trademark of AT&T.

STM

Purpose	Returns the data in the named stream.
Syntax	`STM=`*`name`*
Input Parameters	`PARAMNAME` *`name`* the name of the stream to return
Response	`Stream,`*`name/length:data`* `PARAMNAME` *`name`* the name of the stream `INT` *`length`* the length of the following data `STREAM` *`data`* the stream data
Example	⇒ `STM=\05Operation 000001` ⇐ `Stream,\05Operation 000001/253:(253 bytes of data)`
Notes	Many stream names contain characters which must be indicated using the "escape" character. For example, most stream names start with \05. Note that the FlashPix Specification uses "\005" to indicate octal representation, and this document uses "\05" to indicate hex representation of the CTL character ASCII 5. The server may generate a FPX_REQUEST_INVALID error indicating that the stream could not be returned. The client may then need to fall back to the basic image commands in order to fetch data from the FlashPix file.

2.4 Composed Image Commands

The commands in this section are optionally supported by a server.

These commands are intended for applications in which server-side processing is particularly valuable. Implementation of these commands will impose significant resource demands on a server beyond those required by the baseline commands.

Conceptually, the server must perform four steps to produce a composed image:

1. The source image is mapped through the Flashpix viewing transforms found in the Flashpix file, producing the file result image.
2. The file result image is mapped through the IIP specified transforms, producing the IIP transformed result image. The IIP transforms must applied in the following order:
   - `FTR` first
   - `CTW` any time after `FTR`
   - `CNT` directly after `CTW`
   - `ROI` before `AFN`
   - `RAR` any time after `ROI`
   - `RGN` after all modifiers except `RFM`, `WID`, and `HEI`
   - `WID`, `HEI`, after `RGN`
   - `RFM` after `WID` and `HEI`
   - `ICC` before `CNT` and `CTW`.
3. The final result image is compressed as requested. The image header is then applied and the resulting data is transmitted as appropriate.

Any repeating command modifier appearing later in sequence replaces any earlier occurances.

Practically speaking, there are interactions between certain transforms which must be taken into account in a real implementation. One such implementation would appear as follows:

1. Combine the image manipulation parameters specified using IIP (FTR, CTW, CNT, ICC, AFN, ROI, RAR, and RGN) with the Flashpix viewing parameters found in the file. This will produce a viewing transform that maps the source image to the destination image. The parameters are combined as follows:
    - The IIP ROI replaces the file ROI
    - The IIP RAR replaces the file RAR
    - The IIP FTR is algebraically added to the file FTR.
    - The IIP AFN is postmultiplied with the file AFN to form a new combined AFN.
    - The IIP CTW is postmultiplied with the file CTW to form a new combined CTW.
    - The IIP ICC replaces any input color space conversion suggested by the Flashpix ICC.
    - RGN is applied either as a separate crop to the spatial transform specified by the combined affine, or combined into the affine
    - WID and HEI are used to select the most appropriate resolution from the Flashpix file and convert the resolution independent transformation to a resolution dependent transformation.
2. The image data is processed through the combined transformation to produce an image of width WID and height HEI.
3. The RFM is applied to the image, producing the final result image.
4. The final result image is compressed as requested and returned.

The Flashpix Implementation guide contains more information on these transforms.

CVT

Purpose	Request an image or a rectangular portion of an image to be returned to the client as a complete transformed image. The image will be created on the server and will have the results of any other transformations (those defined within the FlashPix file and those requested by the client) applied in the proper sequence.
Syntax	CVT=*format*
Input Parameters	NAME *format* the format in which to return the image. May be JPEG or FPX in version 0.9 of this protocol.
Response	a complete image with the appropriate MIME type header applied.
Example	CVT=JPEG
Notes	The following commands precede the CVT command and are used to modify the behavior of CVT: FTR, CTW, CNT, QLT, CIN, ICC, AFN, ROI, RAR, RGN, WID, HEI, RFM and RST. Unlike the other commands in this document which are applied as they appear, these command modifiers have a pre-defined processing order as defined above. Multiple instances of a CVT command in a single HTTP request are not supported. Monochrome or YCC colorspace images are always returned as RGB when the format is JPEG (JFIF).

2.5 Composed Image Command Modifiers

The commands in this section modify the compose image commands. The ability of the server to perform these commands is indicated by the FPX-server capabilities field.

The Composed Image Command Modifiers are parsed from left to right or receipt order, but are executed (applied) in the pre-defined order defined above. Modifiers appearing later replace previously appearing modifiers. The RST command can be used to reset all modifiers to their defaults.[should we use an example?]

Composition of images containing opacity information are composed over an opaque white background.

RGN

Purpose Define the region of the image to be returned. This region is specified in resolution-independent coordinates and specifies the region after transformation.

Syntax `RGN=`*`left`*`,`*`top`*`,`*`width`*`,`*`height`*

Input Parameters

`FLOAT` *`left`*
the left coordinate of the image.

`FLOAT` *`top`*
the top coordinate of the image.

`FLOAT` *`width`*
the width of the image.

`FLOAT` *`height`*
the height of the image.

Response

Example `RGN=0,0,1.3,.2`

Notes If the `RGN` command is not specified, the entire image at its maximum resolution is returned. Values outside the bounds of the image are valid. Transparent areas of the result image are assumed to be black.

`RGN` specifies what portion of the output coordinate space is to be returned. This allows clients to fetch a desired region of the image. `RGN` does not specify the result image, just which portion of the result coordinate space to return.

If `WID` and `HEI` specify a size which does not match the aspect ratio, `RGN` is scaled anamorphically.

FTR

Purpose Specify the filtering value viewing parameter.

Syntax `FTR=`*`filter`*

Input Parameters

`FLOAT` *`filter`*
the filtering value

Response

Example `FTR=-15.5`

Notes `FTR` sharpens or blurs the image depending on the value chosen. A value of 0 neither sharpens nor blurs. Values greater than 0 sharpen, while values smaller than 0 blur. See section 7.2.2 of the FlashPix Specification version 1.0 for more information.

CTW

Purpose Specify the color-twist viewing parameter

Syntax `CTW=`A_{11}`,`A_{12}`...,`A_{34}`,`A_{44}

Input Parameters

`FLOAT` A_{11}`..`A_{44}
the values for the matrix. The matrix is 4×4 as follows:

$$\begin{matrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{matrix}$$

Response

Example `CTW=1.1,0,0,0,0,1.1,0,0,0,0,1,1,0,0,0,0,1`

Notes

CNT

Purpose	Specify the image contrast adjustment
Syntax	`CNT=contrast`
Input Parameters	`FLOAT` *`contrast`* the contrast value
Response	
Example	`CNT=.9`
Notes	CNT may be used for contrast adjustment. A value of 1.0 indicates no contrast change. See 7.2.5 FlashPixSpecification version 1.0.

QLT

Purpose	Specify the JPEG ‘Q’ factor to use for JPEG images
Syntax	`QLT=quality`
Input Parameters	`INT` *`quality`* A quality factor between 0 and 100
Response	
Example	`QLT=95`
Notes	This is the 'Q' factor of the JPEG compression applied after image composition and all other transforms. The definition of this Q factor is implementation specific.

CIN

Purpose	Specify the compression group index number to use when creating a composed image via the `CVT` command
Syntax	`CIN=index`
Input Parameters	`INT` *`index`* the index of the compression group to use.
Response	
Example	`CIN=2`
Notes	If no `CIN` is specified, the server may choose a compression group as appropriate.

ICC

Purpose	Specify the ICC profile
Syntax	`ICC=`*`length,data`*
Input Parameters	`INT` *`length`* the length of the following ICC profile data `STREAM` *`data`* the ICC profile stream
Response	
Example	
Notes	

AFN

Purpose: Specify an Affine Transform to apply to the image

Syntax: AFN=A_{11},A_{12}...,A_{43},A_{44}

Input Parameters: FLOAT A_{11}...A_{44}

the values for the matrix. The matrix is 4×4 as follows:

$$\begin{matrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{matrix}$$

Response

Example: AFN=0.86,-0.49,0,0.35,0.49,0.86,0,-0.3,0,0,1,0,0,0,0,1

Notes

ROI

Purpose: Specify the region of interest in resolution-independent coordinates.

Syntax: `ROI=`*`left,top,width,height`*

Input Parameters:

`FLOAT` *`left`*
the left coordinate of the region. Valid values are between 0 and Rs, the aspect ratio of the source image.

`FLOAT` *`top`*
the top coordinate of the region. Valid values are between 0 and 1.

`FLOAT` *`width`*
the width of the region. Valid values are between 0 and Rs, the aspect ratio of the source image.

`FLOAT` *`height`*
the height of the region. Valid values are between 0 and 1.

Response

Example: `ROI=.1,.23,1.2,.56`

Notes: The `ROI` specifies a rectangle of non-transparent image data. Applying the `ROI` is like doing the following steps in Photoshop:

1. Create a new image the same size as the source image. This image will represent the `ROI`.
2. Fill the entire image with transparent (black). This represents the portion of the source image that is outside the `ROI`.
3. Fill the portion of the image that is inside the `ROI` with white.
4. Multiply the new and the source images together.

Neither the size nor the aspect ratio of the image have changed. Conceptually, the "whole image" is still the same size, but part of it is transparent.

RAR

Purpose: Specify the result aspect ratio

Syntax: `RAR=ratio`

Input Parameters:

`FLOAT` *`ratio`*
The ratio of the image width to its height.

Response

Example: `RAR=1.5`

Notes: The result image is defined to be that portion of the output coordinate space with a top left corner at (0,0) and a bottom right corner at (RAR,1). If `RAR` isn't set the bottom right corner is (Rs,1), where Rs is the source aspect ratio. Pixels outside this result rectangle are to be considered transparent.
`RAR` must be positive and non-zero.
Note: Rs indicated the source image. See ROI for more information.

RST

Purpose: Resets the modifiers of the composed image command.

Syntax: `RST=`*`modifier[*,modifier]`*

Input Parameters:

`COMMAND` *`modifier`*
the modifier to be reset.

Response

Example: `RST=AFN`

Notes: The affected modifiers are those in Section 2.5 (minus RST itself). The modifier "*" indicates all modifiers are reset.

RFM

Purpose	Rotate the image by 90, 180, or 270 degrees counterclockwise and/or flip (mirror) about horizontal or vertical axis. Rotation is performed first, then Flip.
Syntax	`RFM=rot[,flip]`

Input Parameters	`INT rot`
	the desired rotation; restricted to 0, 90, 180, or 270.
	`INT flip`
	the desired mirror axis; restricted to 0 (horizontal) and 90 (vertical). None if not present.

Response	
Example	`RFM=90`
Notes	The axes are derived from mathematics conventions and not clocks or maps, hence 0 is horizontal and rotation is counterclockwise.

WID

Purpose	Specify the width in pixels of the composed image.
Syntax	`WID=width`
Input Parameters	`INT width` the desired width in pixels.
Response	
Example	`WID=100`
Notes	If `WID` is used without an `HEI` modifier, the height of the image is determined by the aspect ratio. Legal `WID` values must be 1 or greater.

HEI

Purpose	Specify the height in pixels of the composed image.
Syntax	`HEI=height`
Input Parameters	`INT height` the desired height in pixels.
Response	
Example	`HEI=200`
Notes	If `HEI` is used without a `WID` modifier, the width of the image is determined by the aspect ratio. Legal `HEI` values must be 1 or greater.

2.6 Object Label Definitions

The following table shows the named objects referenced in both the client request and server response. Multiple values returned by an object are separated by a single space. Returned labels and values are always preceded by the object label.

FPX-version

Purpose	describes the major and minor version of this protocol
Syntax	`FPX-version`
Input Parameters	
Response	`FPX-version:`*`vers`* `FLOAT` *`vers`* the revision of the protocol implementation.
Example	⇒`OBJ=FPX-version` ⇐`FPX-version:0.9`
Notes	This is returned with all responses.

FPX-server

Purpose	describes the vendor ID and server capabilities
Syntax	`FPX-server`
Input Parameters	
Response	`FPX-server:`*`vendor`* `1*(`*`.capabs)`* `INT` *`vendor`* the ID of the vendor. `INT` *`capabs`* one or more bitmasks describing the capabilities of the server.
Example	⇒`OBJ=FPX-server` ⇐`FPX-server:999.7`
Notes	The first parameter of `FPX-server` object indicates the server extension vendor ID. Vendor ID's assigned currently are listed in Annex 1. Vendor ID 255 is reserved for "unregistered vendor." Vendor ID 999 is reserved for "experimental", either may be used by anyone. The second parameter for the FPX-Server object indicates the server capabilities. This integer value is a bit-masked flag value, and the bit designations are specified in section 2.8.

FPX-socket

Purpose: describes the address and port for socket based communications

Syntax: `FPX-socket`

Input Parameters:

Response:

`FPX-socket:`*`a.b.c.d port flag url`*

`OCTET` *`a..d`*
the IP address of the server with which to connect

`INT` *`port`*
the port number to use

`INT` *`flag`*
if non-zero, the specified server is not the same as the originating server.

`*UCHAR` *`url`*
the URL of the server (containing the CGI, if appropriate) to which to send subsequent commands

Example:

⇒`OBJ=FPX-socket`

⇐`FPX-socket:207.88.16.75 8080 1 http://fpx.server.com/cgi-bin/fpxservercgi`

Notes: Unless configured to specifically omit this object, a server will always return this information within a request for `Basic-info`. Although a server may not itself support socket based connections, this object may be returned to provide for rendezvous with alternate servers.
This object is never returned via a socket connection.

Basic-info

Purpose: a convenience meta-object which returns basic server and image information. This label returns the following objects:

`FPX-version`	`FPX-server`	`[FPX-socket]`
`File-classID`	`Maxsize`	`Resolution-number`
`Colorspace,*,*`		`View-info`

Syntax: `Basic-info`

Input Parameters:

Response: refer to each label.

Example:

Notes:

View-info

Purpose: a convenience meta-object which returns viewing parameter information. This label returns the following objects:

`Filtering-value`	`Color-twist`	`Contrast-adjust`
`ROI`	`Affine-transform`	`Aspect-ratio`

Syntax: `View-info`

Input Parameters:

Response: refer to each label.

Example:

Notes: If a `View-info` object contains a default value, it need not be returned. Default values are defined for each View-info object in the Flashpix Specification.

Summary-info

Purpose: a convenience meta-object which returns image information. This label returns the following objects:

`Title`	`Subject`	`Author`
`Keywords`	`Comments`	`Last-author`
`Rev-number`	`Last-save-dtm`	`Edit-time`
`Last-printed`	`Create-dtm`	`App-name`

Syntax: `Summary-info`

Input Parameters:

Response: refer to each label.

Example:

Notes:

Max-size

Purpose: describes the maximum width and height of the image.

Syntax: `Max-size`

Input Parameters:

Response:

`Max-size:`*`width height`*

`INT` *`width`*

the width in pixels of the image at the highest resolution

`INT` *`height`*

the height in pixels of the image at the highest resolution

Example:

⇒`OBJ=Max-size`

⇐`Max-size:1024 768`

Notes:

Resolution-number

Purpose: describes the number of resolutions available in the FlashPix file.

Syntax: `Resolution-number`

Input Parameters:

Response:

`Resolution-number:`*`res`*

`INT` *`res`*

the number, indexed from 1, of available resolutions in the FlashPix file

Example:

⇒`OBJ=Resolution-number`

⇐`Resolution-number:7`

Notes:

Colorspace

Purpose
describe the colorspace and channel assignment for the image or optional subimage.

Syntax
`Colorspace,`*`res`*`[,`*`sub`*`]`

Input Parameters
`RANGE` *`res`*
the resolution or range of resolutions to return colorspace information for

`RANGE` *`sub`*
the subimage or range of subimages to return colorspace information for

Response
`Colorspace,`*`res,subim:clb pmo colspc numpla plane`* `*[` *`plane`*`]`
`RANGE` *`res`*
the resolution number of the image for which the colorspace is specified

`INT` *`subImage`*
the subimage of the image for which the colorspace is specified

`INT {0, 1}` *`clb`*
indicates whether the colorspace is calibrated

INT {0, 1} *pmo*indicates whether the opacity has been premultiplied

`INT` *`colorspace`*
the colorspace information

`INT` *`numPla`*
the number of planes in the image

`INT` *`plane`*
color description for each plane

Example
```
⇒OBJ=Colorspace,0-3,1
⇐Colorspace,0,1:0 0 2 3 0 1 2
⇐Colorspace,1-3,1:0 0 3 3 0 1 2
```

Notes
The color spaces are enumerated in detail in section III 3.1.5.2 of the FlashPix Specification version 1.0. If subimage is not specified the default is the primary subimage. If ranges are specified the number of colorspaces returned is the count of the resolution range times the count of the subimage range. All resolutions are returned for each subimage in sequence.
The example describes an image in which the base resolution is PhotoYCC and the lower resolutions (1-3) are NRGB.

ROI

Purpose: describe the region of interest.

Syntax: `ROI`

Input Parameters:

Response:

`ROI:`*`left top width height`*
`FLOAT` *`left`*
the left coordinate of the region.

`FLOAT` *`top`*
the top coordinate of the region.

`FLOAT` *`width`*
the width of the region.

`FLOAT` *`height`*
the height of the region.

Example:

⇒`OBJ=ROI`
⇐`ROI:.5 .101 1.2 .745`

Notes:

Affine-transform

Purpose: describe the affine transform.

Syntax: `Affine-transform`

Input Parameters:

Response:

`Affine-transform:`A_{11} A_{12} A_{13} ... A_{42} A_{43} A_{44}
`FLOAT` $A_{11}...A_{44}$
the values for the matrix. The matrix is 4×4 as follows:

$$\begin{matrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{matrix}$$

Example:

⇒`OBJ=Affine-transform`
⇐`Affine-transform:.86 -.49 0 .35 .49 .86 0 -.3 0 0 1 0 0 0 0 1`

Notes:

Aspect-ratio

Purpose: describe the result aspect ratio viewing parameter.

Syntax: `Aspect-ratio`

Input Parameters:

Response:

`Aspect-ratio:`*`ratio`*
`FLOAT` *`ratio`*
the aspect ratio of the image

Example:

⇒`OBJ=Aspect-ratio`
⇐`Aspect-ratio:1.5`

Notes: The aspect ratio is the ratio of image width to image height.

Filtering-value

Purpose: describe the filtering value viewing parameter.
Syntax: `Filtering-value`
Input Parameters:
Response:
`Filtering-value:`*`value`*
`FLOAT` *`value`*
the filtering value for the image
Example:
⇒`OBJ=Filtering-value`
⇐`Filtering-value:9.8`
Notes:

Color-Twist

Purpose: describe the color twist matrix.
Syntax: `Color-twist`
Input Parameters:
Response:
`Color-twist:`A_{11} A_{12} A_{13} `...`A_{42} A_{43} A_{44}
`FLOAT` $A_{11}...A_{44}$
the values for the matrix. The matrix is 4×4 as follows:

$$\begin{matrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{matrix}$$

Example:
⇒`OBJ=Color-twist`
⇐`Color-twist:1.1,0,0,0,0,1.1,0,0,0,0,1.1,0,0,0,0,1`
Notes:

Contrast-adjust

Purpose: describe the contrast adjustment value.
Syntax: `Contrast-adjust`
Input Parameters:
Response:
`Contrast-adjust:`*`value`*
`FLOAT` *`value`*
the contrast adjustment value for the image
Example:
⇒`OBJ=Contrast-adjust`
⇐`Contrast-adjust:0.5`
Notes: A value of 1.0 indicates no contrast change. See 7.2.5 Flashpix v1.0.

Comp-group

Purpose: The compression headers for the type and indices specified.

Syntax: `Comp-group,`*`type`*`,`*`index`*

Input Parameters:

`INT` *`type`*
the type of compression group to return

`RANGE` *`index`*
the group's index or range or indices to return

Response: `Comp-group,`*`type`*`,`*`index`*`/`*`length`*`:`*`data`*

`INT` *`type`*
the type of compression group

`RANGE` *`index`*
the index of the compression group

`INT` *`length`*
the length of the following data

`STREAM` *`data`*
the compression group

Example:

⇒`OBJ=Comp-group,2,*`

⇐`Comp-group,2,0/562:<binary>`

Notes:
Compression groups may have non-sequential indices from 1-255.
See section 4.1.2 of the FlashPix Specification version 1.0 for compression types. See table 3.9 in the FlashPix Specification V1.0 for a description of the JPEG abbreviated header table.
Note the single color compression type encodes the color returned as a four byte field (returned as 8HEX) in the compression subtype, with null tile data. (see command TIL, above).
The example shows a request for all JPEG compression tables.
Several groups containing identical data may be returned using a range for the index.

Title

Purpose: The title of the FlashPix image.

Syntax: `Title`

Input Parameters:

Response: `Title/`*`length`*`:`*`string`*

`INT` *`length`*
the length of the following data

`STREAM` *`string`*
the title data

Example:

⇒`OBJ=Title`

⇐`Title/8:The Moon`

Notes:

Subject

Purpose: The subject of the FlashPix image.

Syntax:
```
Subject
```

Input Parameters:

Response:
```
Subject/length:string
INT length
```
the length of the following data
```
STREAM string
```
the title data

Example:
```
⇒OBJ=Subject
⇐Subject/13:Lunar Eclipse
```

Notes:

Author

Purpose: The author of the FlashPix image.

Syntax:
```
Author
```

Input Parameters:

Response:
```
Author/length:string
INT length
```
the length of the following data
```
STREAM string
```
the title data

Example:
```
⇒OBJ=Author
⇐Author/11:Buck Rogers
```

Notes:

Keywords

Purpose: The keywords of the FlashPix image.

Syntax:
```
Keywords
```

Input Parameters:

Response:
```
Keywords/length:string
INT length
```
the length of the following data
```
STREAM string
```
the title data

Example:
```
⇒OBJ=Keywords
⇐Keywords/9:Astronomy
```

Notes:

Comment

Purpose: The comment of the FlashPix image.
Syntax: `Comment`
Input Parameters:
Response:
`Comment/`*`length`*`:`*`string`*
`INT` *`length`*
the length of the following data
`STREAM` *`string`*
the title data
Example:
⇒`OBJ=Comments`
⇐`Comments/0:`
Notes: This example shows correct response for an empty or missing object.

Last-author

Purpose: The last author of the FlashPix image.
Syntax: `Last-author`
Input Parameters:
Response:
`Last-author/`*`length`*`:`*`string`*
`INT` *`length`*
the length of the following data
`STREAM` *`string`*
the title data
Example:
⇒`OBJ=Last-author`
⇐`Last-author/11:Buck Rogers`
Notes:

Rev-number

Purpose: The revision number of the FlashPix image.
Syntax: `Rev-number`
Input Parameters:
Response:
`Rev-number/`*`length`*`:`*`string`*
`INT` *`length`*
the length of the following data
`STREAM` *`string`*
the title data
Example:
⇒`OBJ=Rev-number`
⇐`Rev-number/3:5.0`
Notes:

Edit-time

Purpose: The edit time of the FlashPix image.
Syntax: `Edit-time`
Input Parameters:

Response:
`Edit-time:`*`date`*
`DATE date`
the editing time

Example:
⇒`OBJ=Edit-time`
⇐`Edit-time:Wed, 26 Jun 1996 14:50:39 -0700`

Notes:

Last-printed

Purpose: The Last-printed time of the FlashPix image.
Syntax: `Last-printed`
Input Parameters:

Response:
`Last-printed:`*`date`*
`DATE date`
the time of last printing

Example:
⇒`OBJ=Last-printed`
⇐`Last-printed:Wed, 26 Jun 1996 15:01:11 -0700`

Notes:

Create-dtm

Purpose: The creation time of the FlashPix image.
Syntax: `Create-dtm`
Input Parameters:

Response:
`Create-dtm:`*`date`*
`DATE date`
the creation date and time

Example:
⇒`OBJ=Create-dtm`
⇐`Create-dtm:Wed, 26 Jun 1996 14:50:39 -0700`

Notes:

Last-save-dtm

Purpose: The time of last save of the FlashPix image.
Syntax: `Last-save-dtm`
Input Parameters:

Response:
`Last-save-dtm:`*`date`*
`DATE date`
the time of last save

Example:
⇒`OBJ=Last-save-dtm`
⇐`Last-save-dtm:Wed, 26 Jun 1996 14:50:39 -0700`

Notes:

App-name

Purpose: The name of the authoring application for the FlashPix image.

Syntax:
```
App-name
```

Input Parameters:

Response:
```
App-name/length:string
INT length
```
the length of the following data
```
STREAM string
```
the name of the authoring application

Example:
```
⇒OBJ=App-name
⇐App-name/15:Picture Perfect
```

Notes:

ICC-profile

Purpose: The ICC profile(s) specified

Syntax:
```
ICC-profile
```

Input Parameters:

Response:
```
ICC-profile /length:data
INT length
```
the length of the following data
```
STREAM string
```
the title data

Example:
```
⇒OBJ=ICC-profile
⇐need good ICC profile example
```

Notes: If no `ICC` profile exists, data length 0 with no data is returned.

File-class-id

Purpose: The Class-ID of the FlashPix file

Syntax:
```
File-class-id
```

Input Parameters:

Response:
```
File-class-id:id
CLASSID id
```
the ID of the file

Example:
```
⇒OBJ=File-class-id
⇐File-class-id:{56616700-c154-11ce-8553-00aa00a1f95b}
```

Notes: Returns the “clipboard format” field of the OLE Structured Storage in the comp-obj stream. If the file does not have a comp-obj stream or if the “clipboard format” field is not set, the server should return an apporopriate class ID for the type of the specified file.

Security

Purpose
Indicates whether each tile in the specified resolution level(s) is locked or unlocked

Syntax
`Security,res`

Input Parameters
`RANGE` *`res`*
the resolution or range of resolutions for which to retrieve indices

Response
`Security,res/length:`*`map`*
`INT` *`res`*
the resolution level

`INT` *`length`*
the length of the following data

`STREAM` *`map`*
the bitmap of secured tiles

Example
⇒`OBJ=Security,*`
⇐`Security,2/2:<bitmap>`
⇐`Security,1/1:<bitmap>`
⇐`Security,0/0:`

Notes
The format of the return value is a bitmap padded to the nearest byte boundary. If a bit is '1' then the corresponding tile is locked. The least significant bit in the first byte of the bitmap corresponds to tile number '0' of resolution level n. The most significant bit in the first byte corresponds to tile number '7', etc.
The security object identifies those tiles which are locked and require additional authorization. No mechanism is specified for authorizing tile access and it is presumed that this is handled external to the IIP. If a request is made for a security object for which all tiles are unlocked, then the return value is null. If multiple resolution layers are completely unlocked, a resolution range can be specified. For example, if the first five levels are unlocked, a valid response is:
Security;0-2/0:
In the example, the lowest resolution level (containing only one tile) has no security restriction. Equivalently, one null byte could be sent. Resolution level 1 and 2 have security bitmaps. For resolution level 1, there are a maximum of 4 tiles in this example, implying that a single byte is sent in which the lower four bits are used. For resolution level 2, there are up to 16 tiles requiring two bytes for a bitmap.

2.7 Command Summary

2.7.1 Basic Commands

Command	Purpose	Syntax
FIF	Specifies the name of the FlashPix image file to work with.	FIF=path
OBJ	Request an object from the server.	OBJ=object
TIL	Request one or more tiles from the server in their native FlashPix format.	TIL=res,tile[,sub]
JTL	Retrieve a tile as a complete JFIF image.	JTL=res,tile[,sub]

2.7.2 Complex Image Commands

Command	Purpose	Syntax
PTY	Retrieve the data for the indicated Property	PTY=*name*,*propertya*[-*propertyb*]
GRP	Gets the 'Rendering Paths' for the image. The Rendering Paths describe the nodal relationship between the objects that are necessary to render the image(s). Only the relationships matching the specified mask are returned.	GRP=*mask*
SDS	Sets the data object store. Subsequent commands will treat the store pointed to by this command as the root storage.	SDS=*doid*[,*doid*]
STM	Returns the data in the named stream.	STM=*name*

2.7.3 Composed Image Commands (optional)

Command	Purpose	Syntax
CVT	Request an image or a rectangular portion of an image to be returned to the client as a complete composed image. The returned image will be rendered on the server and will have the results of any other transformations applied.	CVT=*format*

2.7.4 Composed Image command modifiers (optional)

Command	Purpose	Syntax
RGN	Define the region of the image to be returned by the CVT command. This region is specified in resolution-independent coordinates and specifies the region after transformation.	RGN=*height*[*left*,*top*,*width*,*height*]
FTR	Specify the filtering value viewing parameter. See section 7.1.4 of the FlashPix Specification version 1.0 for more information.	FTR=*filter*
CTW	Specify the color-twist viewing parameter	CTW=A_{11},A_{12}...,A_{43},A_{44}
CNT	Specify the image contrast adjustment	CNT=contrast
QLT	Specify the JPEG 'Q' factor to use for JPEG images	QLT=quality

CIN	Specify the compression group index number to use	CIN=index
ICC	Specify the ICC profile	ICC=*length,data*
AFN	Specify an Affine Transform to apply to the image	AFN=$A_{11},A_{12}...,A_{43},A_{44}$
ROI	Specify the region of interest in resolution-independent coordinates	ROI=*left,top,width,height*
RAR	Specify the result aspect ratio	RAR=ratio
RST	Resets the modifiers of the image command	RST=*modifier[*,modifier]*

2.8 General Composed Image Command Modifiers

Command	Purpose	Syntax
RFM	Rotate the image by 90, 180, or 270 degrees counterclockwise and/or flip (mirror) about horizontal or vertical axis	RFM=*rot[,flip]*
WID	Specify the width in pixels of the composed image	WID=*width*
HEI	Specify the height in pixels of the composed image	HEI=*height*

2.9 Server Capability Table

A server may optionally provide capabilities outside the baseline Internet Imaging Protocol capabilities. The availability of these capabilities is indicated through the integer in the FPX-server capabilities field. The following capabilities are defined:

Capability	BIT	Description
CVT-JPEG	1	Return a single-level stitched JPEG image.
CVT-FPX	2	Return the FlashPix file.
CVT-MJPEG	3	Return a JPEG image with image transformations applied
CVT-MFPX	4	Return a FlashPix image with image transformations (see above) applied. This bit indicates the capability of the server to process the image and return a new data stream representing a file containing the processed image.

For example, the integer 4 or 5 indicates the server can return a JPEG image using a compression group selected by the client.

2.10 Examples

```
Client Request:
                FIF=Moon.fpx&OBJ=Basic-info&OBJ=Comp-group,*&OBJ=Title
Server Response:
                FPX-version:1.0CRLF
                FPX-server:0.0CRLF
                File-class-ID:{56616700-c154-11ce-8553-00aa00a1f95b}CRLF
                Max-size:1000 1000CRLF
                Resolution-number:5CRLF
                Colorspace,0-4,0:0 0 3 3 0 1 2CRLF


                ROI:0 0 1.5 1.CRLF
                Affine-transform:.86 -.49 0 .35 .49 .86 0 -.3 0 0 1 0 0 0
                0 1CRLF
                Aspect-ratio:1.5CRLF
                Filtering-value:0CRLF
                Color-twist:1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1CRLF
```



```
                    Contrast-adjust:1.0CRLF
                    Comp-group,2,0/785:<785 byte data stream>CRLF
                    Title/19:the moon in the skyCRLF
Client Request:
                    FIF=Moon.fpx&OBJ=Comp-group;4&TIL=2,44&TIL=3,0-1
Server Response:
                    FPX-version:1.0CRLF
                    Comp-group;2,4/547:<547 byte data stream>CRLF
                    Til,2,0,44/12296:<12296 byte data stream>CRLF
                    Til,3,0,0/980:4 <978 byte data stream>CRLF
                    Til,3,0,1/1011:4 <1009 byte data stream>CRLF
Client Request:
                    FIF=Moon.fpx&PTY=56616500%2DC154%2D11CE%2D8553%2D00AA00A1F
                    95B,22000000-22000005
Server Response:
                    FPX-version:1.0CRLF
                    Property,56616500-C154-11CE-8553-
                    00AA00A1F95B,22000000/33:<data stream>CRLF
                    Property,56616500-C154-11CE-8553-
                    00AA00A1F95B,22000001/121:<data stream>CRLF
                    …etc.…
```

2.10.1 Examples of CVT request

```
Client Request:
                    FIF=moon.fpx&RGN=150,0,0,.3,.2&CVT=JPEG
Server Response:
                    Content-type:image/jpeg
                    <JPEG Data Stream>
Client Request:
                    FIF=Moon.fpx&CVT=FPX
Server Response:
                    Content-type:image/fpx
                    <FPX Data Stream>
```

2.11 Error Handling

A special label “Error” indicates that an error occurred. If the error is fatal, then the server could not complete the request. The associated return value includes an informative error message, if possible. If incompatible or unsupported features are specified, then the server returns a response of MIME type “application/netfpx” and a single fatal error. The first parameter after the label indicates the severity of the error. Valid values are “FPX_FATAL” and “FPX_WARNING”. The second parameter is the error code. Error codes are listed below. The syntax of error responses is:

```
Error:(FPX_FATAL|FPX_WARNING) ERRCODE [error message optional]
```

The stream is terminated following a FPX_FATAL error. FPX_WARNING implies that the server is able to complete the request, but a condition was raised during processing which may require the client’s attention. For example:

```
FPX-Version:0.1
Title/8:My Image
Resolution-number:2
Error:FPX_FATAL 3 FlashPix Corrupted File.
```

ERRCODES are integers from a list that includes the following error codes:
[We need to add a section to specify when to use these errors?]

ERRORS	CODES
FPX_OK	0

FPX_INVALID_FORMAT_ERROR	1
FPX_FILE_WRITE_ERROR	2
FPX_FILE_READ_ERROR	3
FPX_FILE_NOT_FOUND	4
FPX_COLOR_CONVERSION_ERROR	5
FPX_SERVER_INIT_ERROR	6
FPX_LOW_MEMORY_ERROR	7
FPX_IMAGE_TOO_BIG_ERROR	8
FPX_INVALID_COMPRESSION_ERROR	9
FPX_INVALID_RESOLUTION	10
FPX_TOO_MANY_LINES	12
FPX_BAD_COORDINATES	13
FPX_FILE_SYSTEM_FULL	14
FPX_MISSING_TABLE	15
FPX_ERROR	19
FPX_UNIMPLEMENTED_FUNCTION	20
FPX_INVALID_JPEG_TABLE	22
FPX_ILLEGAL_JPEG_ID	23
FPX_MEMORY_ALLOCATION_FAILED	24
FPX_OBJECT_CREATION_FAILED	26
FPX_INVALID_TILE	29
FPX_CONVERSION_TYPE_INVALID	30
FPX_REQUEST_INVALID	31
FPX_W_COORDINATES_OUT_OF_RANGE	1000

3. HTTP Protocol Implications

- Requests are in the form of Key – Value pairs. Multiple instances of identical keys may be present in a request.
- When multiple requests are received, they are processed left-to-right. Multiple requests are delimited by "&".

3.1 MIME type

The MIME type[4] associated with the Internet Imaging Protocol is:

application/netfpx

The MIME type associated with the FlashPix image format is:

image/fpx

It is important to make the distinction between the application/netfpx MIME type which is associated with the IIP and image/fpx MIME type which is associated with the FlashPix file format. If the server includes a IIP server module, it will return the MIME type application/netfpx when the client application access FlashPix file on the server. Local FlashPix files or complete remote FlashPix files can be accessed with the MIME type image/fpx. It is also possible for a properly configured IIP server to return an image type such as JPEG upon request, or to serve image tiles originating from other image formats.

3.2 Syntax and Notation

Client requests are in the standard "name/value" form of application/x-www-form-urlencoded. When this protocol is used with HTTP, client requests can be sent using either the "GET" or "POST" methods. An example of proper client request syntax for a CGI server implementation called FPXR is:

```
GET http://server/address/FPXR?FIF=Moon.fpx&OBJ=Resolution-number
```

3.3 URL Construction

There are two types of World Wide Web Internet Imaging Protocol servers anticipated: integrated server extensions and CGI (an external server process). In these examples, a CGI process named FPXR is called.

With an integrated server extension, the name of the image is implied from the URL. A URL of the form

```
http://address/dir1/dir2/file.fpx
```

implies the additional parameter to the client of

```
FIF=dir1/dir2/file.fpx
```

Given the sample client request of

```
FIF=file.fpx&OBJ=basic-info
```

the URL constructed to perform this client request to a CGI server would be

```
http://address/path/FPXR?FIF=file.fpx
```

An example request of

[4] The image/fpx MIME type and application/netfpx MIME type are registered with IANA.


```
FIF=dir1/dir2/file.fpx&OBJ=resolution-count&OBJ=security;*
```

can be expressed as

```
CGI:     http://address/path/FPXR?FIF=dir1/dir2/file.fpx?OBJ=r esolution-
         count&OBJ=security,*
Integr:  http://address/dir1/dir2/file.fpx?OBJ=resolution-count&OBJ=security,*
```


4. Sockets Protocol Implications

Low level socket based connections are possible with this protocol. Socket based connections are state based, are in many cases faster, and allow for connections to be maintained for the duration of a session. Socket based connections, however, may prove difficult or impossible in certain network environments such as those implementing firewalls.

Neither servers nor clients are required to support socket based connections, but they must support the appropriate informational and fall-back protocol elements to assure compatibility. Assuming the client has a method for obtaining address and port information, initial HTTP connections are not necessary.

There are a few syntax implications in the socket based connection environment. Specifically, there is an expanded character set available, removing the need to escape characters outside the 7-bit ASCII range. Additionally, commands in a socket based connection should never be concatenated on a single line – each command must be sent as an individual line with a CRLF pair as a terminator.

State is maintained in a socket based connection for the duration of the connection.

If a query returns no data, an acknowledge (“OK” CRLF) is returned. Completion of a response by the server is signified by an end ("END" CRLF).

4.1 Establishing a Sockets connection from an HTTP connection

The initial connection of a client to a server may be made via an HTTP based connection. In this case, assuming that the server is capable of a socket based connection, the server will provide the FPX-socket object during the first transaction cycle, conveying information which describes how to initiate a socket based connection. If the client chooses to take advantage of the socket based connection model, it may then open a socket based connection to the IP address and port number provided by the FPX-socket object.

The socket based server and the HTTP based server do not need to be the same. It is possible, and in certain situations desired, to have the initial server connection only provide *rendezvous* information pointing to a second alternative server.

4.2 Reversion from Socket to HTTP

Certain environments may not be capable of supporting a socket based connection as described above, but this deficiency can not be known *a-priori*. To allow for the possible advantages of a socket based connection, a second fall back may be necessary when it is not possible.

If a client attempts a socket based connection which times out, it can then resume an alternative connection via HTTP.

The time-out time should be chosen by the client carefully as to avoid long delays as seen by the user. A possible method for determining the time-out may be to select it to equal 2 times the round-trip packet delay between client and server, which can be easily directly measured.

4.3 Sockets Examples

Below is an example scenario depicting the establishment of a socket based connection.

```
->
http://www.somewhere.com/cgi-bin/server?FIF=picture.fpx
<-
FPX-version:0.9CRLF
FPX-server:1 31CRLF
FPX-socket:10.1.1.1 9999 0CRLF
```


File-class-ID:{56616700-c154-11ce-8553-00aa00a1f95b}CRLF
Maxsize:3096 2048CRLF
Resolution-number:6CRLF
Colorspace,0-5,0:0 0 3 3 0 1 2CRLF
Filtering-value:1.0CRLF
Color-twist:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0CRLF
Contrast-adjust:1.2CRLF
ROI:0 0 1 1.5CRLF
Affine-transform:1 1 1 1 1 1 1 1 1 1 1 1 1 1 1CRLF
Aspect-ratio:1.5CRLF
The client then opens a connection to address 10.1.1.1, port 9999 and requests tiles:->
TIL=1,0-1CRLF
<-
Tile,1,0,0/253:*data*CRLF
Tile,1,0,1/745:*data*CRLF
->
OBJ=Comp-group,2,1
<-
Comp-group2,1/75:*data*CRLF

5. Annex 1 Vendor ID's

A server vendor may choose to be identified through the vendor ID. The following vendor ID's are assigned. Assignments can be registered by making a request to the editors of this document. 255 and 999 vendor ID values may be used without restriction or registration.

ID	Vendor
0	Hewlett-Packard Company
1	Live Picture, Incorporated
2	Eastman Kodak Company
255	unregistered vendor
999	experimental server

6. Credits and contact information

Authors names are presented in alphabetical order

- Laurent Albert lalbert@livepicture.com
- David Kulp dkulp@livepicture.com
- Andrew Mutz mutz@hplabs.hp.com
- Robert Phipps rmp@kodak.com
- Marc Spencer mspencer@kodak.com

The authors wish to acknowledge the contributions of the members of the Flashpix consortium, the extended teams working in each company, and in particular the work of Howard Dworsky, Andy Fitzhugh, J. Scott Houchin, Ho John Lee, Ricardo Motta, Steve Shaffer, and Gerrie Shults.

This document represents work not yet finalized. Comments, discussion and change requests may be directed to the email list iip@hplaef.hpl.hp.com. This list may be subscribed to by sending an email message to majordomo@hplaef.hpl.hp.com with message body “subscribe iip”.

