This page intentionally left blank.

Internet Imaging Protocol

Version 1.0.5

You may E-mail questions and comments to the IIP mail-list at:

```
iip@hplaef.cup.hp.com
```

To subscribe to the IIP mail-list, send an E-mail mesage to:

```
majordomo@hplaef.cup.hp.com
```

with the words

```
subscribe iip
```

in the body of the message.

Contents

SECTION 2 Reference 21

SECTION 1

Design Specification

1.1 Overview

The Internet Imaging Protocol (IIP) is designed to communicate tiled image data and related descriptive information over network connections. 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

- Support a rich image rendering model
- Leverage existing network environments
- Create an interoperable framework
- Optimize network utilization

1.1.1 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[1], 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.

IIP can be thought of as presenting a client with a virtual FlashPix file over a network. The FlashPix image format provides a convenient storage format and a powerful image rendering model. Alternately, IIP may be thought of as a uniform method for presenting an image from any format to a client in a resolution-independent, bandwidth-efficient manner.

Images stored in FlashPix format are organized into square 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 best resolution 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.

The protocol is designed to be transport protocol layer independent. Specific guidelines for implementing the protocol over HTTP and low-level TCP/IP sockets are in Section 1.2.

---

1. See FlashPix Format Specification 1.0 available at http://www.kodak.com/go/FlashPix

   The *FlashPix*™ format is defined in a specification and a test suite, developed and published by *Eastman Kodak Company* in collaboration with *Hewlett-Packard Company, Live Picture Inc.* and *Microsoft Corporation*. Only products that meet the specification and pass the test suite may use the *FlashPix* file format name.

1.2 Transport Protocol Specification

IIP is designed to work within existing transports such as HTTP and direct socket connections. For example, requests are concatenated using “&” for HTTP (Common Gateway Interface style), but are separated by CRLF for sockets connections.

Internet Imaging Protocol communication is composed of requests to the server (commands) and the specification of data structures (objects) which are to be passed over the communications channel.

1.2.1 HTTP Protocol Usage

HTTP requests are composed of one or more commands expressed as key-value pairs. Multiple instances of identical keys may be present in a request.

When multiple commands are received, they are processed left-to-right. Multiple commands are delimited by “&”.

Responses are composed of one or more objects. Multiple objects are separated by CRLF.

1.2.1.1 MIME type

The MIME type[2] associated with the Internet Imaging Protocol response is:

- application/vnd.netfpx

The MIME type associated with the FlashPix image format is:

- image/vnd.fpx

It is important to make the distinction between the MIME type associated with the IIP and the MIME type associated with the FlashPix file format. If a server includes an IIP server module, it will return the MIME type application/vnd.netfpx when a client accesses the server. Local FlashPix files or complete remote FlashPix files can be accessed with the MIME type image/vnd.fpx.

2. The image/vnd.fpx MIME type and application/vnd.netfpx MIME type are registered with IANA.

In the cases where the server's response may require more than one MIME type (the CVT and JTL commands) the precidence of the MIME types is:

1. Image (image/jpeg or image/vnd.fpx)
2. Error (application/vnd.netfpx)
3. IIP Object (application/vnd.netfpx)

1.2.1.2 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.

1.2.1.3 URL Construction

There are two types of World Wide Web Internet Imaging Protocol servers anticipated: integrated server extensions and CGI (an external server process) servers.

With an integrated server extension, the name of the image is implied from the URL and replaces the FIF command.

For example, an integrated server extension serves a URL of the form

```
http://address/dir1/dir2/file.fpx?OBJ=IIP,1.0&TIL=4,*
```

in a manner equivalent to a CGI server, FPXR, receiving a request to a URL,

```
http://address/FPXR?FIF=/dir1/dir2/file.fpx
   &OBJ=IIP,1.0&TIL=4,*
```

1.2.1.4 HTTP headers and caching

It is possible to cache responses to IIP requests in proxy caches if information about the validity of a resource is available. Servers should construct responses including the Last-Modified HTTP header whenever possible.

1.2.2 Sockets Protocol Usage

Low level socket based connections are possible with this protocol. Socket based connections maintain state, 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 CRLF as the 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).

1.2.2.1 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 `IIP-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 `IIP-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.

1.2.2.2 Reversion from Socket to HTTP

Certain environments may not be capable of supporting a socket-based connection as described above, but this deficiency may not be known *a-priori*.

If a client attempts and fails a socket-based connection, it can open a connection via HTTP.

The time-out time should be chosen by the client carefully 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.

1.3 Command Overview

1.3.1 Basic Commands (required)

command	Purpose	Syntax
FIF	Specifies the name of the image resource	FIF=*path*
OBJ	Request an object from the server.	OBJ=*object*
TIL	Request one or more tiles from the server in their native coded format.	TIL=*res*,*tile*[,*sub*]
SDS	Sets the data object store. Subsequent commands within a request will treat the store pointed to by this command as the root storage. See Section 2.1.1.	SDS=*doid* *(,*doid*)

1.3.2 Optional Commands

These commands are optionally supported by a server. Support can be discovered via the capability bitmap returned in the `IIP-server` object. See Section 3.3 for more information regarding server capability bits.

command	Purpose	Syntax
CVT	Request 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*
JTL	Retrieve a tile as a complete JFIF image.	JTL=*res*,*tile*[,*sub*]

1.3.2.1 Transform composed Image command modifiers (optional)

These commands modify the CVT command; alone, they have no function. CVT may be implemented by a server without supporting these modifiers. Support for these modifiers can be discovered via the capability bitmap returned in the `IIP-server` object. See Section 3.3 for more information regarding server capability bits.

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]*

1.3.2.2 General Composed Image Command Modifiers (optional)

These commands modify the CVT command; alone, they have no function. CVT may be implemented by a server without supporting these modifiers. Support for these modifiers can be discovered via the capability bitmap returned in the IIP-server object. See Section 3.3 for more information regarding server capability bits.

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*

1.4 Object Overview

Object	Purpose	Syntax
`IIP`	describe the major and minor version of the protocol	`IIP,version`
`IIP-server`	describe the vendor ID and server capabilities	`IIP-server`
`Basic-info`	MetaObject. This returns basic server and image information	`Basic-info`
`View-info`	MetaObject. This returns viewing parameter information	`View-info`
`Max-size`	describe the maximum width and height of the image	`Max-size`
`Resolution-number`	describe the number of resolutions available	`Resolution-number`
`Colorspace`	describe the color space and channel assignment for the image and optional sub-images.	`Colorspace,`*`res`*`,[`*`sub`*`]`
`Comp-group`	describe the compression headers for the type of compression and indices specified.	`Comp-group,`*`type`*`,`*`index`*
`ROI`	describe the region of interest in resolution independent coordinates.	`ROI`
`Affine-transform`	describe the affine transformation matrix	`Affine-transform`
`Aspect-ratio`	describe the aspect ratio	`Aspect-ratio`
`Contrast-adjust`	describe the contract adjustment	`Contrast-adjust`
`Filtering-value`	describe the filtering value	`Filtering-value`
`Color-twist`	describe the color twist matrix	`Color-twist`
`File-class-id`	describe the class-id	`File-class-id`
`IIP-socket`	describes the URI for rendezvous with a socket-based server	`IIP-socket`
`Summary-info`	MetaObject. This returns image description information	`Summary-info`
`Copyright`	copyright information	`Copyright`
`Title`	describes the title	`Title`
`Subject`	the subject	`Subject`
`Author`	the author	`Author`
`Keywords`	the keywords	`Keywords`
`Comment`	the comment	`Comment`
`Last-author`	the last author	`Last-author`
`Rev-number`	the last revision	`Rev-number`
`Edit-time`	the edit time	`Edit-time`
`Last-printed`	the last printing time	`Last-printed`
`Create-dtm`	the creation date and time	`Create-dtm`
`Last-save-dtm`	the last save date and time	`Last-save-dtm`
`App-name`	the name of the last application to edit the image	`App-name`
`ICC-profile`	the ICC profile	`ICC-profile`
`Property`	the data for the indicated FlashPix Property	`Property,`*`name`*`,`*`propertya`*`[-`*`propertyb`*`]`

`Render-path`	Meta-object. The ‘Rendering Path’ for the image. The Rendering Path describes the nodal relationship between the objects that are necessary to render the image(s). Only the relationships matching the specified mask are returned.	`Render-path,mask`
`Stream`	the entire FlashPix stream	`Stream,name`
`Security`	Security information for the image	`Security`
`IIP-opt-comm`	describe the list of vendor specific additional commands	`IIP-opt-comm`
`IIP-opt-obj`	describe the list of vendor specific additional objects	`IIP-opt-obj`

1.5 Security Overview

IIP provides access to image and image-related data via IIP commands and requests for IIP objects. It is foreseeable that certain resources should be protected from general access over a network. Although IIP does not itself enforce a security policy, the protocol does provide a means for a server to inform a client that a request has failed, and to provide additional information regarding the cause of the failure.

The capability to traverse structured storage entities enables significant functionality within FlashPix files. A result of this functionality is a potential security hole for files accessible to the IIP server unless a security policy is implemented.

Unqualified requests for Stream objects represent a prominent security concern. Stream objects should be secured by class-id, to prohibit access to any stream within a file of any type.

Possible restriction policies include:

- File by File-class-id
- Object by name
- Object by file
- Property by file
- Stream objects by stream name.

There are 4 layers of resources in the IIP architecture: Server, File, Command, and Object. The server may secure instances of these resources as appropriate.

On an image by image basis, commands and objects have provisions for security. Queries to resources which fail can be responded to with 4 types of messages: No access, unsupported, not found, or malformed request. Information regarding the availability of a resource for a particular function cannot be determined prior to access (with the exception of tiles, which may have an associated access bitmap.) When an access fails due to authorization failure, the secured state of a resource is returned via an error code.

1.5.1 Example Security Policies

Although specific security implementations are outside the scope of this protocol, it is important to illustrate some of the possible security policies that can be implemented using the IIP.

Two examples of security policy are restriction by inclusion and restriction by exclusion. Restriction by inclusion limits access to a known list of objects. This model is easily administered; an administrator only permits globally acceptable data entities to be served. This model implies the access table must be updated if a new extension is to be supported.

Another alternative is to design a server which allows global access to image resources, with specific resources listed as excluded from the allowed set. This method provides the same level of security as the previous model for a known set of image resources. It is far easier to manage a table of constrained access, but this method presents a higher security risk; access to unknown resources in a file are always granted.

1.6 Error Handling

A special object label `Error` indicates that the server could not complete a requested action. The error message contains information regarding why the error occurred.

Errors occur when accessing a resource – a file, object, or command.

Each error falls into one of four groups: Badly formed or otherwise un-parsable requests, unavailable resources, failed authorization for a requested resource, or unsupported resource on the server. Additional server-centric errors also exist to signal when a fault outside of the above domain occurs.

If a request for image data is denied, an error message is returned. Four error classes are defined:

4. File
5. Command
6. Object
7. Server

In addition, four error numbers within each class are defined:

1. Syntax error (for example, object name is misspelled.)
2. Unsupported (for example, function not supported for this image, or on this server.)
3. Unavailable (for example, the object does not exist in the image)
4. Permission denied (for example, an object is access-restricted)

The error class and number are combined into a 2-digit error code by appending the number to the class, separated by a single space. The server error-class utilizes error codes listed in the table below.

	Malformed/ Syntax	Unsupported	Unavailable	Permission Denied
File	1 1	1 2	1 3	1 4
Command	2 1	2 2	2 3	2 4
Object	3 1	3 2	3 3	3 4
Server	4 n see table below			

ERRORS	CODES
CONVERSION_ERROR	4 1
PERMISSION_DENIED	4 2
MEMORY_ALLOCATION_FAILED	4 3
OBJECT_CREATION_FAILED	4 4
FILE_WRITE_ERROR	4 5
FILE_READ_ERROR	4 6
Vendor-defined error	4 7

Error

Purpose

describe an error condition.

Response

`Error/`*`length`*`:`*`errclass errnum object`* [*`message`*]

`INT` *`length`*

The length of the error response

`INT` *`errclass`*

The error class

`INT` *`errnum`*

The error code

`LABEL` *`object`*

The object (including any parameters) that was requested, if applicable, or “none” if none

`STREAM` *`message`*

An optional, less than 240 character unicode string containing additional error message information

Example

```
⇒OBJ=IDONTEXIST
⇐Error/49:3 3 IDONTEXIST Object not found.CRLF

⇒OBJ=VIEW-INFO
⇐Colorspace,0-4,0:0 0 3 3 0 1 2CRLF
⇐ROI:0 0 1.5 1.CRLF
⇐Affine-transform:0.86 -0.49 0 0.35 0.49 0.86 0 -0.3
   0 0 1 0 0 0 0 1CRLF
⇐Error/19:3 3 Filtering-valueCRLF
⇐Error/15:3 3 Color-twistCRLF
⇐Error/19:3 3 Contrast-adjustCRLF
⇐Error/16:3 3 Aspect-ratioCRLF

⇒OBJ=Stream,%05Operation%200000001
⇐Error/122:3 4 Stream,%05Operation%200000001 Stream
   requests not served on this web site.CRLF
```

Notes

As in all IIP strings, the error message is represented in unicode.
The length of an error includes all data between the colon (`:`) and the `CRLF` pair.

SECTION 2

Reference

2.1 Request and Response Syntax

2.1.1 Basic Rules

The following basic rules are used throughout this specification:

- Commands and Labels are case-insensitive.
- 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.
- IIP is stateless between requests. Requests are composed of one or more commands. Requests are delimited within the transport. (For example, by the end of an HTTP GET request, or the closing of a socket-based connection) [1]
- IIP commands are parsed sequentially. (For example, commands are parsed left to right in the case of CGI, and in receipt order for sockets). The objects in the response need not be in the order requested. Certain optional commands may not be executed sequentially; these commands have a pre-defined execution sequence.
- All string data[2] (Title, Author, etc...) is expressed as Unicode. (In this document, the inter-character nulls for ASCII strings have been omitted)

1. State management methods could be applied to maintain state between requests. HTTP Keep-alive, session handles, or cookies are examples of such methods. Since it is not possible to define a single appropriate method, at this time, any implementation of a persistance method is implementation specific.

2.1.2 Global Syntax

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. Object labels are not string data

2.1.3 General Syntax for Requests

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

```
COMMAND     = 3ALPHA
REQUEST     = COMMAND "=" 1*UCHAR
OLABEL      = NAME *("," RANGE | PARAMNAME)
NAME        = 3ALPHA *(DIGIT | ALPHA | "-")
PATH        = <URI as per IETF RFC 1945 encoded with MIME
               type application/x-www-form-urlencoded >
PARAMNAME   = 1*(alpha | DIGIT | ESCAPE | SAFE)
UCHAR       = UNRESERVED | ESCAPE
UNRESERVED  = ALPHA | DIGIT | SAFE | EXTRA
ESCAPE      = "%" HEX HEX
SAFE        = "$" | "-" | "_" | "." | "+"
EXTRA       = "!" | "*" | "'" | "|" | "(" | ")" | ","
RANGE       = "*" | INT | CONTRANGE
CONTRANGE   = INT "-" INT
INT         = 1 * 10 DIGIT
FLOAT       = ["-"] ( INT | 1*16 DIGIT "." 1*16 DIGIT )
STREAM      = * OCTET
PID         = 8HEX
```

- The value of `INT` may be between 0 and 2^{31}-1, or 2,147,483,647
- The value of the left `INT` in a `CONTRANGE` must be less-than or equal-to the value of the right `INT` in a `CONTRANGE`
- In `RANGE`, `"*"` should not be translated into a literal minimum and maximum range but rather to indicate a request for all existing objects. If no objects are present, a single error is returned. Otherwise, only the existing objects are returned.

2.1.4 General Syntax for Responses

Responses to requests for image data are sent from a server to a client and are composed of objects. An objects consists of a label and data associated with the label. Multiple objects can be concatenated as described in Section 2.1 and Section 2.2.

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)
LENGTH      = INT
CLASSID     = "{" 8HEX "-" 4HEX "-" 4HEX "-" 4HEX "-" 12HEX
               "}"
DATE        = <as per RFC 1123>
RESERVED    = ";" | "/" | "?" | ":" | "@" | "&" | "="
PARAMNAME   = 1*(ALPHA | DIGIT | ESCAPE | SAFE)
NAME        = 3ALPHA *(DIGIT | ALPHA | "-")
STREAM      = * OCTET
PID         = 8HEX
ACKNOWLEDGE = "OK" CRLF <sockets implementation only>
END         = "END" CRLF <sockets implementation only>
```


2.2 Command Reference

Basic requests are composed of these commands, which must be supported by all IIP servers.

FIF

Purpose Specify the name of the image resource

Syntax `FIF=path`

Input Parameters

`PATH` *`path`*
Path (generally relative) to the image file. The x-www-form-urlencoded MIME type is used to allow paths with SP, etc.

Response `FIF` returns nothing when accompanied by other commands.

If `FIF` and `OBJ=IIP` are the sole key-value pairs, the server responds as if it had received an OBJ=IIP and an OBJ=Basic-info command.

Example

⇒ `FIF=/myimage/finepicture.fpx`

Notes The server defines a `root directory` for images accessed using IIP. Paths are relative to this root directory; `FIF=finepicture.fpx` is equivalent to `FIF=/finepicture.fpx`

This path is often set to the same document root path as the HTTP server.

This key-value pair is present in an IIP request unless the server has native support for the IIP protocol for a specific image; in that case the path to the image resource is already given by the URL.

Since objects are returned in no specific order, requests under HTTP containing multiple FIFs, while not illegal, will return data which is not uniquely source identified.

OBJ

Purpose Request an object from the server

Syntax `OBJ`=object

Input Parameters

`OLABEL object`
The label of a requested data object. See Section 2.3

Response Varies by object

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 64 × 64 pixel tiles from the server in their native coded format

Syntax

`TIL=res,tile[,sub]`

Input Parameters

`RANGE` *`res`*
The desired resolution

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

`RANGE` *`sub`*
The desired sub-image; defaults to the primary sub-image

Response

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

`INT` *`tile`*
Tile number

`INT` *`sub`*
Sub-image number of tile; defaults to the primary sub-image (zero)

`INT` *`length`*
Length of data

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

Example

⇒ `TIL=4,0-5` (See notes below regarding tile ranges)
⇐ `Tile,4,0,0/<len>:<data>`
⇐ `Tile,4,1,0/<len>:<data>`
⇐ `Tile,4,4,0/<len>:<data>`
⇐ `Tile,4,5,0/<len>:<data>`

Notes

As in the example above, a request for a range of tiles is responded to with multiple tile entities. A range of sub-images or resolutions can be used.

Access to sub-images other than the primary sub-image is provided to permit access to resources beyond the FlashPix 1.0 Specification, such as FlashPix files with extensions.

If SDS is supported for the image,(see below) the sub-images or ranges are returned from the primary object store by default, or from the object store indicated using SDS; see the FlashPix specification 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:

0	1	2	3
4	5	6	7
8	9	10	11

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

For the above example, a request for tiles 3-10 returns tiles 2, 3, 6, 7, 10, and 11.

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™ `cd` command.

The ‘path’ specified is always defined from the root of the Structured Storage structure (absolute).

Access to storages other than the root storage is provided to permit access to resources beyond the FlashPix 1.0 Specification, such as FlashPix files with extensions.

SDS provides access to information with potentially open scope. See Section 2.5 for additional information regarding image resource security.

The reception of a subsequent `FIF` command will reset SDS to the root data storage.

2.2.1 Optional Commands

Optional commands may be supported by servers. Clients may support the information returned by optional commands. A client may discover optional commands by requesting the `IIP-opt-comm` object. The IIP-opt-comm and IIP-opt-obj objects indicate only those commands and objects additionally supported by a server beyond those defined in this specification.

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 sub-image. If omitted, this defaults to the primary sub-image

Response A complete JFIF image of that tile

Example

⇒ `JTL=4,0`
⇐ `<data>`

Notes A range of tiles may not be requested.

Images are returned as MIME type image/jpeg. In sockets communication, no MIME header is required. 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.

2.2.1.1 Composed Image Commands

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 three steps to produce a composed image:

- The source image is mapped through the FlashPix viewing transforms found in the FlashPix file, producing the file result image.
- The file result image is mapped through the IIP specified transforms, producing the IIP transformed result image. The IIP transforms must be 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
- 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 occurrences.

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:

- 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 CNT is algebraically multiplied to the file CNT
    - The IIP AFN is post-multiplied with the file AFN to form a new combined AFN
    - $AFN_{new} = AFN_{file} * AFN_{IIP}$
    - The IIP CTW is pre-multiplied with the file CTW to form a new combined CTW
    - $CTW_{new} = CTW_{iip} * CTW_{file}$
    - 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
- The image data is processed through the combined transformation to produce an image of width WID and height HEI
- The RFM is applied to the image, producing the final result image
- The final result image is compressed as requested and returned

The FlashPix Implementation guide contains more information on these transforms.

Editor's Note: There will be examples added here to illustrate CVT modifier combination

---

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` (MIME type image/jpeg) or `FPX` (MIME type image/vnd.fpx) in version 1.0 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).

There are additional levels of functionality which can be optionally supported by the server (see server capability table, Section 3.3) to apply client specified transformations in addition to the file-based ones.

2.2.1.2 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 `IIP-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.

Servers may choose from several levels of Composed Image Modifier Command support as outlined in Section 3.3. If a request is made containing a modifier that is not supported by the server, an Error object with error class 2, error number 2 (command/unsupported) should be returned in lieu of the requested image. Clients can determine what capabilities are available via the IIP-server object.

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

To illustrate the persistance of the Command Modifiers, the request sequence:

```
⇒ FIF=foo.fpxCRLF
⇒ OBJ=IIP,1.0CRLF
⇒ WID=240CRLF
⇒ CVT=JPEGCRLF
⇐ (Image)
⇒ FIF=bar.fpxCRLF
⇒ CVT=JPEGCRLF
⇐ (Image)
```

will first select image foo.fpx and return a JFIF image scaled to a width of 240 pixels, and a height calculated to preserve the aspect ratio. The server should continue to process and return a JFIF image bar.fpx also scaled to a width of 240.

Since this persistence is cumulative, it is desirable to reset the modifiers. In this case:

```
FIF=foo.fpx&OBJ=IIP,1.0&WID=240&FIF=bar.fpx&
  RST=WID&HEI=120&CVT=JPEG
```

the RST command will reset the previous WID setting, allowing the width of image bar.fpx to be calculated from the aspect ratio. Without the RST present, the image would be anamorphically scaled to exactly 240×120.

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=0,0,1.3,0.2`

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=`$A_{11},A_{12},\ldots,A_{34},A_{44}$

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=1.1,0,0,0,0,1.1,0,0,0,0,1,0,0,0,0,1`

Notes

CNT

Purpose Specify the image contrast adjustment

Syntax `CNT=contrast`

Input Parameters `FLOAT` *contrast*

The contrast value

Response

Example ⇒ `CNT=0.9`

Notes CNT may be used for contrast adjustment. A value of 1.0 indicates no contrast change. See 7.2.5 FlashPix Specification 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 The use of the ICC CVT Modifier will not be possible using the HTTP GET method.

AFN

Purpose Specify an Affine Transform to apply to the image

Syntax $\texttt{AFN=}A_{11},A_{12},\ldots,A_{34},A_{44}$

Input Parameters $\texttt{FLOAT } A_{11}..A_{44}$
The values for the matrix. The matrix is 4×4as 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=0.1,0.23,1.2,0.56`

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 2.2.1.2 (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 mathematic 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.

There is no default for `WID`.

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.

There is no default for `HEI`.

The following table shows the required objects referenced in both the client request and server response. Multiple values returned by an object are separated by a single space. Objects always begin with the object label.

Editor's Note: There is currently work underway to provide detailed definitions regarding the FlashPix Architecture, including additional information on contrast, color-twist, and sharpening operations.

2.3 Object Reference

IIP

Purpose

Describes the major and minor version of this protocol

Syntax

`IIP,`*`version`*

Input Parameters

`FLOAT` *`versiom`*
The version of the protocol requested by the client

Response

`IIP:`*`version`*
`FLOAT` *`version`*
The revision of the protocol implementation

Example

⇒ `OBJ=IIP,1.0`
⇐ `IIP:1.0`

Notes

This object should be present in all requests, and is returned with any response which follows the opening of a connection to the server. For HTTP, this implies that this object is present within all requests and responses; in sockets, it should be present only in the initial request and response.

`IIP` allows a client to inform a server of the version of this protocol that it implements. The server may, at its option, communicate at the version requested, or specify that it will communicate at a version different (greater or less than) from the client.

If this object is ommitted from a request, an error response (error class 3, error code 1) will be generated by the server.

IIP-server

Purpose Describes the vendor ID and server capabilities

Syntax `IIP-server`

Input Parameters

Response

`IIP-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=IIP-server`
⇐ `IIP-server:999.7`

Notes The first parameter of `IIP-server` object indicates the server extension vendor ID. Vendor ID's assigned currently are listed in Annex 2. 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 `IIP-Server` object indicates the server capabilities. This integer value is a bit-masked flag value, and the bit designations for certain capabilities are specified in Annex 3. The bit fields are expressed as a sequence of little-endian bytes. The least significant 8 bits are reserved for common use, and other bits may be used by vendors for private options.

Basic-info

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

```
IIP-server        Colorspace,*,*
View-info         Max-size          Resolution-number
```

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 must return the following objects or indicate that the objects do not exist via the proper error response:

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

Syntax

`View-info`

Input Parameters

Response

refer to each label.

Example

Notes

Section 1.6 describes the proper error response for requests to non-existant objects. Every request for an object must have a paired response.

Max-size

Purpose

Describes the maximum width and height of the source (pre-transformed) 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 image

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 sub-image

Syntax

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

Input Parameters

`RANGE` *`res`*
The resolution or range of resolutions for which to return colorspace information

`RANGE` *`sub`*
The sub-image or range of sub-images for which to return colorspace information

Response

`Colorspace,`*`res,subim:clb pmo colspc numpla`* `1*(SP` *`plane)`*
`RANGE` *`res`*
The resolution number of the image for which the colorspace is specified

`INT` *`subim`*
The sub-image of the image for which the colorspace is specified, indexed from 0

`INT {0, 1}` *`clb`*
a ‘0’ indicates that the colorspace is calibrated; a ‘1’ indicates that the colorspace is uncalibrated

`INT {0, 1}` *`pmo`*
a ‘1’ indicates that the opacity has been premultiplied; a ‘0’ indicates that the opacity has not 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. NRGB is default. If sub-image is not specified the default is the primary sub-image. If ranges are specified the number of colorspaces returned is the count of the resolution range times the count of the sub-image range.

The example describes an image in which the base resolution is PhotoYCC and the higher resolutions (1-3) are NRGB.

Several groups containing identical data may be returned using a range for the index.

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 of 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. This stream will be null for types 0 and 1

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.

See Section 3.4, Annex 4 Compression Types, for more detailed information.

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:0.5 0.101 1.2 0.745`

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:0.86 -0.49 0 0.35 0.49 0.86 0 -0.3 0 0 1 0 0 0 0 1`

Notes The default matrix of

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

represents a null transform.

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.

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.

Filtering-value

Purpose Describe the filtering value viewing parameter which sharpens or blurs the image

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 A filtering value of 0 represents no sharpening or blurring.

Color-twist

Purpose

Describes the color twist matrix which modifies the tone and colors of the image

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

File-class-id

Purpose

The Class-ID of the image 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 appropriate class ID for the type of the specified file.

IIP-socket

Purpose Describes the address and port for socket based communications

Syntax `IIP-socket`

Input Parameters

Response `IIP-socket:`*URI*
`URI` *URI*
The URI of the server socket. A port number will often be required

Example ⇒ `OBJ=IIP-socket`
⇐ `IIP-socket:207.88.16.75:8080`

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.

Summary-info

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

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

Syntax `Summary-info`

Input Parameters

Response refer to each label.

Example

Notes

Copyright

Purpose

An object which returns copyright information about the image. This should include the owner of the copyright.

Syntax

`Copyright`

Input Parameters

Response

`Copyright/`*`length`*`:`*`string`*
`INT` *`length`*
The length of the following data

`STREAM` *`string`*
The copyright information

Example

⇒ `OBJ=Copyright`
⇐ `Copyright/90:The Forbin Project, 1978, all rights reserved`

Notes

This object corresponds to the Copyright message property in the intellectual property group property set in a FlashPix file.

As stated in the Section 2.1, all strings are represented as Unicode. The inter-character nulls have been removed from the examples in this document.

Title

Purpose

The title of the 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/16:The Moon`

Notes

Subject

Purpose The subject of the image

Syntax `Subject`

Input Parameters

Response `Subject/`*`length`*`:`*`string`*
`INT` *`length`*
The length of the following data

`STREAM` *`string`*
The subject data

Example
⇒ `OBJ=Subject`
⇐ `Subject/26:Lunar Eclipse`

Notes

Author

Purpose The author of the image

Syntax `Author`

Input Parameters

Response `Author/`*`length`*`:`*`string`*
`INT` *`length`*
The length of the following data

`STREAM` *`string`*
The author data

Example
⇒ `OBJ=Author`
⇐ `Author/22:Buck Rogers`

Notes

Keywords

Purpose The keywords of the image

Syntax `Keywords`

Input Parameters

Response `Keywords/`*`length`*`:`*`string`*
`INT` *`length`*
The length of the following data

`STREAM` *`string`*
The keyword data

Example
⇒ `OBJ=Keywords`
⇐ `Keywords/18:Astronomy`

Notes

Comment

Purpose The comment of the image

Syntax `Comment`

Input Parameters

Response `Comment/`*`length`*`:`*`string`*
`INT` *`length`*
The length of the following data

`STREAM` *`string`*
The comment data

Example
⇒ `OBJ=Comment`
⇐ `Comment/0:`

Notes This example shows correct response for an existing object with empty content.

Last-author

Purpose The last author of the image

Syntax `Last-author`

Input Parameters

Response `Last-author/`*`length`*`:`*`string`*

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

`STREAM` *`string`*
The last-author data

Example
⇒ `OBJ=Last-author`
⇐ `Last-author/22:Buck Rogers`

Notes

Rev-number

Purpose The revision number of the image

Syntax `Rev-number`

Input Parameters

Response `Rev-number/`*`length`*`:`*`string`*

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

`STREAM` *`string`*
The revision number data

Example
⇒ `OBJ=Rev-number`
⇐ `Rev-number/3:5.0`

Notes

Edit-time

Purpose The edit time of the 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 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 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 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 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/30:Picture Perfect`

Notes

ICC-profile

Purpose

The ICC profile specified

Syntax

`ICC-profile`

Input Parameters

Response

`ICC-profile/`*`length`*`:`*`data`*
`INT` *`length`*
The length of the following data

`STREAM` *`data`*
The ICC profile data

Example

⇒ `OBJ=ICC-profile`
⇐ `ICC-profile/8048:<data>`

Notes

If no `ICC` profile exists, the appropriate error is returned.

This command should return the ICC profile data only. In FlashPix, this requires the 28 byte header to be stripped.

The ICC (International Color Consortium) profile is a standard for color

processing. This profile defines a map from the color space of the image data to one of the ICC defined interchange color spaces. This allows any system that uses ICC compliant color management to transform the data into any device color space (like monitors or printers) for which a profile exists. For more information on the ICC, ICC profiles, and some color managent scenarios, see the ICC web site, http://www.color.org/.

IIP-opt-comm

Purpose

Retrieve supported optional commands from a server

Syntax

`IIP-opt-comm`

Input Parameters

Response

`IIP-opt-comm:*`*`label`*
`NAME` *`label`*
A space-delimited list of the optional commands supported

Example

⇒ `OBJ=IIP-opt-comm`
⇐ `IIP-opt-comm:CVT`

Notes

Optional capabilities may also be indicated in the `IIP-server` object.

This object is intended to allow server implementations to implement a superset of the commands specified in this document.

IIP-opt-obj

Purpose Retrieve supported optional object labels from a server

Syntax `IIP-opt-obj`

Input Parameters

Response `IIP-opt-obj:*`*`label`*
`NAME` *`label`*
A space-delimited list of the optional objects supported

Example
⇒ `OBJ=IIP-opt-obj`
⇐ `IIP-opt-obj:Camera`

Notes Optional capabilities may also be indicated in the `IIP-server` object. The labels returned are available for the specific image.

This object is intended to allow server implementations to implement a superset of the objects specified in this document.

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.

The security object may only exist for the primary subimage.

2.3.0.1 Complex Image Objects

Complex image objects should be supported by servers supporting FlashPix images. While this data is generally accessible via other objects, a server and client which both implement structured storage can efficiently use these objects. There are security considerations regarding access to structured storage streams.

These objects 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 containing 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 data in 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.

These objects provide for access to resources beyond those defined in the FlashPix 1.0 Specification, such as FlashPix files with extensions.

Property

Purpose

Retrieve the data for the indicated Property

Syntax

`Property,`*`name`*`,`*`propertya`*`[-`*`propertyb`*`]`

Input Parameters

`1*UCHAR` *`name`*
The name of the property set. This name must be represented using x-url-encoding

`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`*
`1*UCHAR` *`name`*
The name 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

⇒ `OBJ=Property,%05Image%20Info,22000000-22000004`
⇐ `Property,%05Image%20Info,22000000/52:`*`data`*`CRLF`
⇐ `Property,%05Image%20Info,22000001/48:`*`data`*`CRLF`
⇐ `Property,%05Image%20Info,22000002/64:`*`data`*`CRLF`
⇐ `Property,%05Image%20Info,22000003/20:`*`data`*`CRLF`
⇐ `Property,%05Image%20Info,22000004/132:`*`data`*`CRLF`

Notes

A range of requests is returned with multiple Property returns.

Property streams are a special case of the more general Structured Storage stream. They are identified by an ASCII Ctrl-E beginning the stream name.

This object contains the Property data directly extracted from the FlashPix file as a SERIALIZEDPROPERTYVALUE structure. No byte-switching or other pre-processing is performed. See Section A.2.1.3 in the FlashPix Specification 1.0

Render-path

Purpose

Gets the 'Rendering Paths' for the image

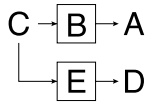
In a FlashPix file, a Rendering path is a set of streams[1] which describe the relationship of an image to its rendering information. A FlashPix image may have multiple rendering paths. The `Render-path` object returns only the rendering paths referred to in the Visible Outputs property in the Global Information property set.

Data within 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 object is intended to allow 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 discontinuous 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).

This object 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, a FlashPix 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. `Render-path` would return A, B and C only.



E is in the rendering path "referred" to indirectly by the visible output (entry for A), but since E is not directly referenced, E should not be returned.

If a FlashPix file names nodes A and D as visible outputs, Render-path would return A, B, C, D, and E.

Syntax

`Render-path,[`*`sub`*`,]`*`mask`*

Input Parameters

`INT` *`sub`*

The sub-image 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

⇒ `OBJ=Render-path,7`
⇐ `Stream,%05Global%20Information/139:`*`data`*`CRLF`
⇐ `Stream,%05Data%20Object%20000001/103:`*`data`*`CRLF`
⇐ `Stream,%05Data%20Object%20000002/143:`*`data`*`CRLF`
⇐ `Stream,%05Data%20Object%20000003/99:`*`data`*`CRLF`
⇐ `Stream,%05Data%20Object%20000004/168:`*`data`*`CRLF`
⇐ `Stream,%05Transform%20000001/145:`*`data`*`CRLF`
⇐ `Stream,%05Transform%20000002/206:`*`data`*`CRLF`

Notes

Multiple objects may be returned for a single `render-path` request. The response is as if one or more stream objects had been requested.

---

1. Streams can contain entire property sets as signified by an ASCII Ctrl-E preceding the stream name

Stream

Purpose

Returns the data in the named stream from the current storage

Syntax

`Stream,`*`name`*`[,`*`byterange`*`]`

Input Parameters

`PARAMNAME` *`name`*
The name of the stream to return

`RANGE` *`byterange`*
A subset of the stream data to return. Byte ranges are indexed from 0, the first byte of the stream

Response

`Stream,`*`name`*`,`*`offset`*`/`*`length`*`:`*`data`*
`PARAMNAME` *`name`*
The name of the stream

`INT` *`offset`*
The offset in bytes from the start of the entire stream that the following returned data contains. If no byterange was specified, this should be 0

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

`STREAM` *`data`*
The stream data

Example

⇒ `OBJ=Stream,%05Operation%20000001`
⇐ `Stream,%05Operation%20000001,0/253:`*`data`*`CRLF`

Notes

Many stream names contain characters which must be indicated using the "escape" character. For example, property-set stream names start with \005. Note that the FlashPix Specification uses "\005" to indicate octal representation, and this document uses "%05" to indicate x-url-encoding representation of the ASCII Ctrl-E character.

The server may generate an error indicating that the stream could not be returned. The client may then fall back to the basic image commands in order to fetch data from the FlashPix file.

2.4 Optional Requests and Objects

2.4.1 Optional Requests

An optional request is composed of at least one optional command. Optional requests may be supported by an IIP server. There are two discovery methods for support of optional commands.

The `IIP-server` object contains a field for identifying capabilities. Public and private capabilities including both commands and objects may be indicated using this field (see Section 1.3 and Section 3.3.)

The `IIP-opt-comm` object may be requested to discover what optional commands are available for a particular image resource. This object consists of a space-delimited list of all optional commands supported. There is no internal mechanism for elaboration of the details of these commands. The vendor ID in the `IIP-server` object disambiguates optional command definitions; each vendor assumes a name space.

A server shall return an error statement when an unsupported command is received. A server may ignore the unsupported portion of an optional request and respond to the supported portion of the request.

Some optional commands are defined in Section 2.2.1.

2.4.2 Optional Objects

Objects embody all non-pixel data from an IIP resource. Optional objects may be associated with image resources. The `IIP-opt-obj` object may be requested to discover what optional objects are available. `IIP-opt-obj` consists of a space-delimited list of all optional object labels supported.

A server shall respond to a request for an unsupported object with an appropriate error response. See Section 1.6.

2.5 Security

Two aspects of security are addressed within IIP: access authentication and access policy. Implementations are not described. A means is provided to inform a client regarding existing tile resources which require authentication. An error response is defined to inform a client regarding authentication failure. Access policies limiting availability of data in IIP addressed files are described.

Security issues in IIP revolve upon two features. First, a content provider may want to provide access to only a portion of an image. Since multi-resolution images are accessed with IIP, this provides a method for allowing clients to preview images at low resolution without allowing access at high resolution. Second, a content provider may have image or non-image data in a file accessed by IIP not intended for open use. Some of the IIP objects could inadvertantly provide access to this information. Some guidance to security concerns is provided.

2.5.1 Access Authentication

Many resources on a network are only available to authenticated users. This specification does not describe means to confirm authentication and such methods are out of scope. The IIP does define the Security object. The Security object forms a bitmap showing the tile-by-tile availability of a resolution layer in an image. Using this object, a client can limit requests to unrestricted tiles if desired. This would minimize challenged or failed requests.

If a restricted resource is requested, the server should authenticate the client before serving the requested data. Failed authentication may be indicated using the IIP error structure or (when appropriate) the HTTP error code responses. The IIP error structure provides a fine granularity of response for failure of a request, but requires the IIP client to implement the error handling facility.

No map of available meta-data is provided.

2.5.2 Access Policy

A server should not restrict access to objects required to view an image (such as Affine-transform) if the tile data is available. A server may restrict access to other metadata such as last-printed (for example).

Access to objects may be defined on any basis appropriate; access on a file name, object name, or command name is possible. Certain particular issues arise when objects referring to structured storage are made.

2.5.3 Direct Structured Storage Requests

IIP defines several objects to provide direct access to structured storage. These objects may be required by applications accessing resources not explicitly defined in IIP but existing on a server. Care should be taken to understand the implications of this access. For example, a client with access to any storage in a structured storage file via the SDS command and Stream object has effective access to any resource in the file which the server can open. This will generally include all data in the file. If a server enables these commands, all data in any file accessible to the IIP server should be considered available to a client. Implementor should be aware that data not normally anticipated by a user (such as old revisions in a document) can be contained in a file and potentially accessed via `SDS` and `Stream` if no policy is implemented.

A reasonable policy in light of this is restriction of these IIP commands to files of certain types, namely FlashPix files. In a structured-storage based system, the class-id of a file may be used to implement such a policy. If access to certain parts of Flashpix files is to be restricted, it may be appropriate to deny access to these files using Stream objects.

All information needed to view a “core” (V1.0) Flashpix file (“core” refers to Flashpix files without extensions) can be obtained without using the `SDS` command or the `Stream`, `Render-path`, and `Property` objects.

2.6 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.

SECTION 3 *Annexes*

3.1 Annex 1: Examples

This section contains several examples of Client-Server communication using IIP. Although the examples here are not exhaustive, they do illustrate a significant number of IIP transactrion cases.

In all examples, the HTTP, and therefore MIME headers are not shown for simplicity.

3.1.1 Basic Example

This example illustrates what may be an initial communication with a server. The client is explicitly asking for `Basic-info`, as well as additionally requesting `Comp-group` and `Title` objects. Note that the server does need to return Basic-info objects or error objects for each object in the `Basic-info` set.

⇒

```
FIF=Moon.fpx&OBJ=IIP,1.0&OBJ=IIP,1.0&
   OBJ=Basic-info&OBJ=Comp-group,2,*&OBJ=Title
```

⇐

```
IIP:1.0CRLF
IIP-server:0.0CRLF
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:0.86 -0.49 0 0.35 0.49 0.86 0 -0.3 0 0 1
   0 0 0 0 1CRLF
Aspect-ratio:1.5CRLF
Error/19:3 3 Filtering-valueCRLF
Error/15:3 3 Color-twistCRLF
Error/19:3 3 Contrast-adjustCRLF
Comp-group,2,0/785:dataCRLF
Title/38:the moon in the skyCRLF
```

The client now requests image tiles: 1 tile from resolution 2, and a range of tiles from resolution 3.

⇒

```
FIF=Moon.fpx&OBJ=IIP,1.0&TIL=2,44&TIL=3,0-1
```

⇐

```
IIP:1.0CRLF
Tile,2,44,0/12296:dataCRLF
Tile,3,0,0/980:dataCRLF
Tile,3,1,0/1011:dataCRLF
```

3.1.2 Complex Object Example

The client is interested in obtaining the copyright information associated with the image. This information is contained in the Image Info property set. Note that property set names must be x-url-encoding encoded.

⇒

```
FIF=Moon.fpx&OBJ=IIP,1.0&PTY=%05Image%20Info,
22000000-22000004
```

⇐

```
IIP:1.0CRLF
Property,%05Image%20Info,22000000/33:dataCRLF
Property,%05Image%20Info,22000001/121:dataCRLF
…etc…
```


3.1.2.1 CVT Request Example

In this example, the client wishes to have the server perform the rendering operation and return the image as a composite JFIF image. Note that the image is returned under its own MIME type.

⇒
```
FIF=moon.fpx&OBJ=IIP,1.0&RGN=150,0,0,0.3,0.2&CVT=JPEG
```

⇐
```
Content-type:image/jpeg
Content-length:495234
<JPEG Data Stream>
```

3.1.3 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&
   OBJ=IIP,1.0
```

⇐
```
IIP:1.00CRLF
IIP-server:1.12CRLF
IIP-socket:10.1.1.1:9999CRLF
Max-size:3072 2048CRLF
Resolution-number:7CRLF
Colorspace,0-5,0:0 0 3 3 0 1 2CRLF
Filtering-value: .0CRLF
Color-twist: 1.0 0 0 0 0 1.0 0 0 0 0 1.0 0 0 0 0 1.0CRLF
Contrast-adjust:1.2CRLF
ROI: 0 0 1.500000 1.0CRLF
Affine-transform: 1.0 0 0 0 0 1.0 0 0 0 0 1.0 0 0 0 0 1.0CRLF
Aspect-ratio:1.500000CRLF
```

The client then opens a connection to address 10.1.1.1, port 9999 and requests tiles. Since the connection is not closed between requests, the `IIP` object is returned only once:

⇒
```
IIP=1.0CRLF
```

⇐
```
IIP:1.00CRLF
OK
END
```



```
⇒
FIF=picture.fpxCRLF

⇐
OK
END

⇒
TIL=2,1-2CRLF

⇐
Tile,2,0,1/2784:dataCRLF
Tile,2,0,2/3157:dataCRLF
END
```

Based on the compression subtype data contained in the tiles returned, the client now requests a Comp-group object:

```
⇒
OBJ=Comp-group,2,1

⇐
Comp-group2,1/574:dataCRLF
END
```


3.2 Annex 2: 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 at the E-mail address on page ii. 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

3.3 Annex 3: 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 `IIP-server` capabilities field. The following capabilities are defined:

Capability	BIT	Description
CVT-JPEG	0	Return a single-level stitched JFIF image MIME type image/jpeg.
CVT-FPX	1	Return the FlashPix file. MIME type image/vnd.fpx.
CVT-MJPEG	2	Return a JFIF image with the Transform composed image command modifiers (see Section 2.2.1.2) 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. MIME type image/jpeg.
CVT-MFPX	3	Return a FlashPix image with the Transform composed image command modifiers (see Section 2.2.1.2) 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. MIME type image/vnd.fpx
CVT-M2JPEG	4	Return a JFIF image with the General composed image command modifiers (see Section 2.2.1.2) applied in combination with the file-based view transforms and other supported transforms. 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. MIME type image/jpeg.
CVT-M2FPX	5	Return a FlashPix image with the General composed image command modifiers (see Section 2.2.1.2) applied in combination with the file-based view transforms and other supported transforms. 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. MIME type image/vnd.fpx
JTL	6	Return a single square tile as a JFIF image.

For example, the integer 21 (binary 010101) indicates the server can return a JPEG image using a compression group selected by the client.

3.4 Annex 4: Compression Types

Prepended to the data returned from a `TIL` command are 8 bytes containing the compression type and subtype information for that tile. The 8 bytes consist of a little-endian 32-bit unsigned INT containing the compression type followed by 4 bytes comprising the compression subtype. This data mirrors the information in the FlashPix tile header fields for compression type and subtype (FlashPix specification 1.0, Section 4.1.2).

The compression Type is a 32 bit unsigned INT, where the following values are defined:

Field Value	Meaning
0x0	Uncompressed data
0x1	Single color compression
0x2	JPEG compression
0xFFFFFFFF	Invalid Tile

For Uncompressed data (compression type 0x0) or Invalid Tile (compression type 0xFFFFFFFF) the compression subtype is unused and must be zero.

For Single color compression (compression type 0x01) the compression subtype contains the actual color of all pixels in the tile. Individual channel values are stored in little-endian format, in the same order and bit-depth as specified by the sub-image color and sub-image numerical format properties, aligned to the 0th bit of the field (see Section 3.1.5.3 of the FlashPix specification, Version 1.0)

The compression subtype information consists of 4 octets. For JPEG compression, the data in these bytes is organized as follows:

First:	Interleave Type
Second:	Chroma Subsampling
Third:	Internal Color Conversion
Fourth:	JPEG Table Selector

Compression types of None, Single-color, and JPEG are the only compression types supported in version 1.0 of IIP. Other compression types may be added in future versions. Compression type values of 0x0 to 0x7FFFFFFF are reserved.

Example: A JPEG compressed, interleaved tile coded with internal color conversion and JPEG table 3 would begin with (in hex) `02 00 00 00  00 22 01 03.`

3.5 Annex 5: Colorspace Definitions

The Colorspace object returns a list of color space parameters for the resolution and sub-image specified. The colorspace information, an integer, has the following interpretation:

Field Value	Meaning
0x0	Colorless
0x1	Monochrome
0x2	PhotoYCC
0x3	NIF RGB

The exact definition for the above color spaces can be found in the FlashPix specification, Version 1.0.

Each colorspace object returns image plane number assignments which map the image planes to color specifications. The mapping is as follows:

Colorspace	Field Value	Meaning
Monochrome	0x0	Monochrome
PhotoYCC	0x0	Luminance
	0x1	Chrominance 1
	0x2	Chrominance 2
NIF RGB	0x0	Red
	0x1	Green
	0x2	Blue
All	0x7FFE	Opacity

Editor's Note: A brief description of the colorspace definitions will be added here.

