IMPLEMENTATION GUIDELINES

BASIC IMAGING PROFILE

Abstract

This document is designed to facilitate the task of implementing the Basic Imaging Profile. It provides recommendations and examples that complement the Basic Imaging Profile specification. It is presented in a Q&A format.

Revision History

Revision	Date	Comments
0.01	1/10/2001	First draft
0.1	4/10/2001	Fujifilm and Toshiba comments included
0.2	12/10/2001	Nokia and Canon comments included
0.4	19/10/2001	Improvements of the BPP related Q&As
0.5	2/11/2001	Changes in the structure of the document
0.6	7/11/2001	Editorial clean-up
0.95	21/12/2001	Version sent to BARB for review
1.0	4/2/2002	Comments from BARB + learning of the Kobe interoperability session added.

Contributors

Contributor	Company
Kenichi Fujii	Canon
Yasuo Fukuda	Canon
Akane Yokota	Canon
Kazuaki Abe	Casio Computer
Ryohei Yamamoto	Casio Computer
Hiroshi Tanaka	Fujifilm
Akinori Yoshioka	Fujifilm
Karl Heubaum	Motorola
Stephane Bouet (owner)	Nokia
Martin Roter	Nokia
Yos uke Tajika	Toshiba
Takuya Kawamura	Toshiba

Role of the Basic Imaging Profile

Q1. Bluetooth 1.1 already includes a File Transfer Profile specification. Why should I implement the Basic Imaging Profile specification? ____ 4

Implementation Requirements

Q2. I already implemented one of the profiles based on the Generic Object Exchange Profile in my product. Can I easily upgrade it to support the Basic Imaging Profile as well? ____ 4

Q3. What is the minimum set of features for a device to qualify as Basic Imaging Profile device? ____ 5

Q4. What are the minimum requirements for the OBEX stack implementation? ____ 6

Q5. What are the minimal requirements for my image encoder/decoder? ____ 7

Basic Imaging Profile and Basic Printing Profile

Q6. Both the Basic Imaging Profile and the Basic Printing Profile propose mechanisms for image printing. Which profileshould I use? ____ 8

Q7. I already implemented the Basic Printing Profile. Can I easily add support for image printing according to the Basic Imaging Profile (or vice-versa)? ____ 8

Security in Basic Imaging

Q8. How is security handled in the Basic Imaging Profile? ____ 14

Q9. Is there a concept of content protection in the Basic Imaging Profile? ____ 14

Implementation details

Q10. What is the role of the ServiceID SDP attribute? How is it used in the profile? ____ 14

Q11. How do I discover/identify a device supporting the Basic Imaging Profile? ____ 15

Q12. Why is the Image Push feature divided into four sub features in the Imaging Responder service record? ____ 16

Q13. How should I use the friendly-name attribute in the Image Properties object? ____ 16

Q14. How is the name attribute used to identify image attachments? ____ 17

Q15. How is the Name header used in a PutImage operation? ____ 17

Q16. What is the recommended usage of the OBEX Abort command? ____ 18

Q17. My implementation cannot handle big data files. I need to negotiate the size of image files in addition to format and pixel size. Is this possible with the Basic Imaging Profile? ____ 18

Q18. Should the Imaging Thumbnail format always be listed as available format in an Image-Properties object? ____ 20

Q19. Can a very simple Image Pull Initiator skip the GetImageProperties function? ____ 20

Q20. What is an empty Image Descriptor? ____ 21

Q21. Can a Remote Display Responder accommodate a Remote Display Initiator that is not supporting the 'Select' command ? ____ 21

Q1. Bluetooth 1.1 already includes a File Transfer Profile specification. Why should I implement the Basic Imaging Profile specification?

The Basic Imaging Profile's primary objective is to enable image format negotiation. It is especially useful when an image exchange involves one or more limited devices (typically embedded devices) that can process only selected image formats. By insuring that images are delivered or retrieved in a format that is usable by the destination, the profile guarantees application level interoperability. In addition, the Basic Imaging Profile provides basic remote control capabilities that are not available in the File Transfer Profile.

When an image exchange involves only high-end devices – devices like PCs for which image formats are of no concern and that do not utilise remote control functions – image files can be treated just like any other file, and therefore can be exchanged using the File Transfer Profile. It is important to note, however, that high-end devices that intend to exchange images with limited devices would have to support the Basic Imaging Profile.

Q2. I already implemented one of the profiles based on the Generic Object Exchange Profile in my product. Can I easily upgrade it to support the Basic Imaging Profile as well?

In principle, it's relatively simple for a device that already supports one of the Generic Object Exchange-based profiles to support the Basic Imaging Profile as well. Indeed, the Basic Imaging profile utilises the same protocol stack as the other Generic Object Exchange-based profiles, thus limiting the extra development work required.

It should be noted, however, that the Basic Imaging Profile places additional constraints on the OBEX implementation compared to the requirements detailed in the Generic Object Exchange Profile (see question 4). This is the result of a difficult trade-off between protocol flexibility and interoperability obligations, given that OBEX in itself does not provide sufficient interoperability guarantees.

Therefore, the answer to this question requires an understanding of the work required to adapt the OBEX stack to the Basic Imaging Profile. If the adaptations are easy – which we expect to be true in the majority of cases – the rest of the effort is limited: introduction of a few additional OBEX headers, user interface modification, and application layer development. If, however, the additional requirements placed on the OBEX implementation are difficult to fulfill with the OBEX stack in use, the implementer is faced with the possibility of integrating a new OBEX stack.

Q3. What is the minimum set of features for a device to qualify as Basic Imaging Profile device?

To qualify as a Basic Imaging Profile compliant device, the capabilities described as mandatory in the profile specification must be supported. Mandatory capabilities are present at several levels, including features, functions, image formats, and image sizes, making a simple, concise answer to this question difficult. The exact set of features a device must support depends on the nature of the device and its usage model. To illustrate the process of customising – in this case, sizing down – a Basic Imaging Profile implementation, we propose an example "James Bond watch" with a built-in camera capable of capturing quarter VGA size images and sending them to another Bluetooth device.

All Basic Imaging Profile compliant devices have a common base called the Generic Imaging capability. Generic Imaging is the ability to either push or pull images (see section 4.1 of the profile specification). No matter how small you want your implementation to be, you must support at least either the Image Push feature or the Image Pull feature. In addition to this Generic Imaging capability, the profile defines a collection of more advanced capabilities that enable limited remote control over another imaging device. Those advanced capabilities are optional and, in the case of our watch, unnecessary.

Each Basic Imaging Profile feature is comprised of a set of functions that perform individual tasks related to that feature. Some functions are mandatory and some are optional. In the case of our watch, the only feature that will be supported is Image Push, which has only two mandatory functions: PutImage and PutLinkedThumbnail (see section 4.3.1). The Image Push feature defines two optional functions: GetCapabilities and PutLinkedAttachment. Omitting these optional functions limits the capabilities of the watch; in particular, omitting the GetCapabilities function means the watch will not detect a possible absence of support for quarter VGA size images on the receiving device, which in turn implies the watch may make one or more attempts to push images that the receiving device cannot use before eventually falling back to the mandatory Imaging Thumbnail format (see question 5).

The structure of the Basic Imaging Profile functions is fixed and mandatory (see the introductory portion of section 4.5), meaning that none of the OBEX fields described in Table 4-21 through Table 4-39 can be omitted. In other words, it is not possible to size down the implementation by dropping OBEX fields. The content of many fields, can, however, be sized down. In the case of our watch, the PutImage function uses Connection ID and Type OBEX headers for which no optimisation is possible. The Name header uses a Unicode string that can be shortened, although more descriptive names are obviously better. Considerable optimisation can be done in the Img-Description header, which carries an Image Descriptor describing the image being pushed. The Image Descriptor includes two required attributes ("encoding" and "pixel") and three optional attributes ("size", "maxsize", and "transformation"). Implementers should carefully consider the trade-off between smaller code and reduced capabilities before deciding to omit an optional attribute.

As a side note, it is interesting to note that the choice of supported XML attributes will also affect how extensive the implementation will have to be in terms of XML encoding and parsing capabilities. This is especially evident for the larger XML DTDs like the Imaging Capabilities and Image Properties objects. If only the mandatory attributes are supported, the need for a full featured XML encoder/parser will be reduced, thus making it possible to adopt an implementation strategy based on pre-coded strings.

The PutLinkedThumbnail function cannot be sized down, since none of the OBEX headers it uses have customisable content.

To sum up, in the case of a watch whose role is to send quarter VGA size images, the minimal implementation will support only the simplest embodiment of the Image Push feature, using only the PutImage and PutLinkedThumbnail functions and the smallest possible Image Descriptor (most likely a pre-coded descriptor). This process can be repeated to narrow down the set of features, functions, and OBEX header contents required for other implementations.

Q4. What are the minimum requirements for the OBEX stack implementation?

Two of the still image working group's major goals had a direct impact on the Basic Imaging Profile's use of OBEX:

1. Keep the implementation as light weight as possible.

2. Minimise the chances of interoperability problems.

To keep implementations compact and robust in the face of interoperability hazards, the profile imposes additional constraints on the OBEX stack. Implementers who plan to use a commercial OBEX stack in their profile implementation should insure that stack meets the following requirements:

- The profile mandates the order in which OBEX headers are delivered. No variation in this order (on a command to command, state to state, or any other basis) is allowed see the first paragraph of section 4.5 in the profile specification).

- The profile defines OBEX response codes and error conditions in greater detail than the OBEX specification. If, for example, the OBEX stack in use offers automated error handling, it must not interfere with or defeat the profile's error mechanisms (see section 5.3 in the profile).

- The profile defines the timing when headers should be sent in case of multi-packets Put or Get responses (see section 5.2.3 in the profile).

- The OBEX stack must gracefully ignore unsupported headers. This is a prerequisite for compatibility with future versions of the Basic Imaging Profile, which may introduce new features and functions that use new OBEX headers. Note that this behaviour also applies to the profile implementation's XML parser, since it may encounter unrecognised elements and attributes introduced in future versions of the profile (see section 5.2 in the profile).

Q5. What are the minimal requirements for my image encoder/decoder?

All Basic Imaging Profile implementations must support the following JPEG-based Imaging Thumbnail format as specified in section 4.4.3 of the profile specification:

- JPEG baseline-compliant
- sRGB as default colour space
- Pixel size: 160x120
- Sampling: YCC422
- One marker segment for each DHT and DQT
- Typical Huffman table
- DCF thumbnail file as file format

The profile cannot guarantee interoperability for the other image encodings it supports: other (non-thumbnail) JPEG variants, BMP, WBMP, PNG, GIF, and JPEG2000. These encodings all offer various options, parameters, and file formats, and it's possible that a decoder incorporated into a Basic Imaging Profile implementation could be presented with an image it cannot process because that image utilises options and parameters or a file format the decoder doesn't understand. The working group decided to adopt this approach rather than mandate a specific set of options and parameters and a file format for each encoding because:

1. There is already a very large collection of image files spread all over the planet, and mandating specific options, parameters, and file formats would inevitably (and unreasonably) exclude a significant portion of that collection.
2. Image encoders and decoders are often implemented in hardware, so mandating options and parameters could limit the adoption of the profile if it forced manufacturers to make hardware changes.

Fortunately, in most cases there exist defacto option, parameter, and file format standards for each image encoding supported by the Basic Imaging Profile; implementations should follow those defacto standards to insure maximum interoperability. The one exception is JPEG2000, since it's relatively new and as of this writing ISO discussions on establishing profiles for different JPEG2000 usage models are still in progress. The working group expects, however, that at the very least a defacto JPEG2000 standard will emerge as this encoding technology is more widely adopted.

Q6. Both the Basic Imaging Profile and the Basic Printing Profile propose mechanisms for image printing. Which profile should I use?

The Basic Imaging and Basic Printing Profiles propose different printing mechanisms because they originated in different markets. The Basic Imaging Profile originated in the digital still camera and dedicated photo printer markets, where the Digital Print Order Format (DPOF) was already established. The Basic Printing Profile originated in the mainstream printer market, so it mandates a page description language (XHTML-Print) and an image encoding standard (JPEG). The still image and printing working groups recognised that interoperability across the profiles is important, so each profile defines a simple baseline printing mechanism that's very similar to its counterpart, making it easy to add support for one profile once the other is implemented (see the following question for details).

Which printing mechanism to use as the "native" mechanism is left to the implementer's discretion. The still image working group expects that cameras will adopt the Basic Imaging Profile's mechanism because they are based on a technology that's already widely implemented in those devices – DPOF. Printers that support both photo and document printing will most likely support both the Basic Imaging Profile and the Basic Printing Profile, although there may be some photo-printing devices that support only the Basic Imaging Profile. Full blown implementations of both profiles in a single device appears unlikely in the short to medium term.

Q7. I already implemented the Basic Printing Profile. Can I easily add support for image printing according to the Basic Imaging Profile (or vice-versa)?

The printing mechanisms proposed by the Basic Printing and Basic Imaging Profiles are slightly different (see the previous question). Both profiles, however, include a baseline version of their optimal printing mechanism that enables images to be pushed and printed one-by-one; these baseline print mechanisms are sufficiently similar that it's easy for one device to support both, providing some interoperability between the profiles. Note, however, that these simplified print mechanisms do not provide user feedback or control, and the resulting output is entirely determined by the printer.

Two scenarios are of particular interest: 1) a Basic Print Profile printer that can accept Basic Image Profile Image Push requests, and 2) a Basic Imaging Profile client that can generate Basic Print Profile simple push requests.

Scenario 1

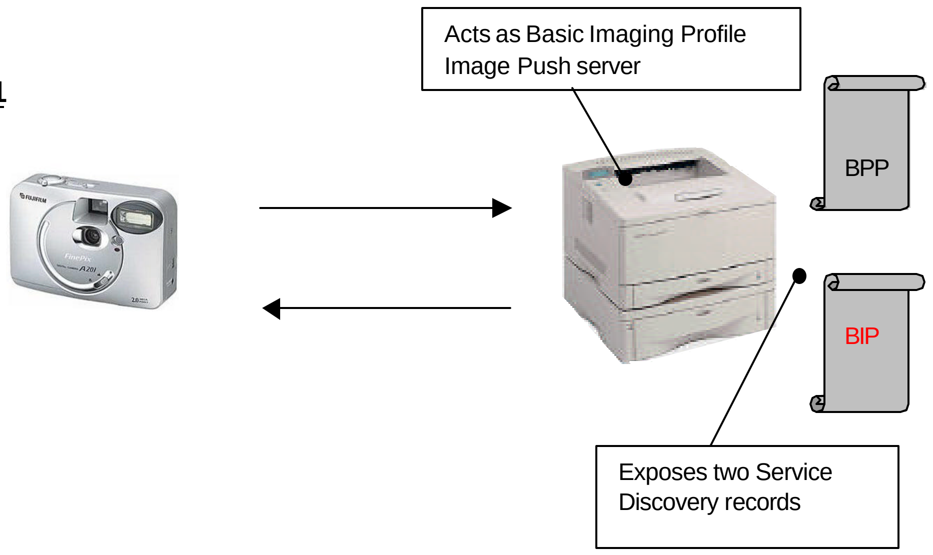
Acts as Basic Imaging Profile Image Push server
BPP
BIP
Exposes two Service Discovery records

*Figure 1: **Scenario 1**. A Basic Printing Profile printer with the ability to receive Basic Imaging Profile Image Push requests advertises this capability via a Basic Imaging Profile Service Discovery record and acts as an Image Push server for the digital still camera.*

Scenario 2

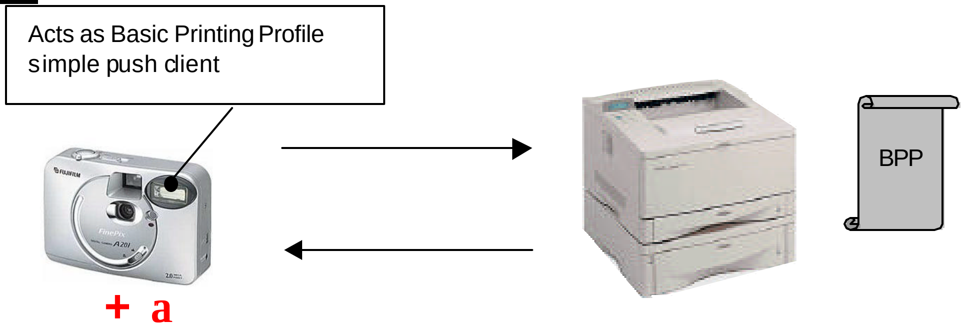

*Figure 2: **Scenario 2.** A Basic Imaging Profile camera has implemented the additional capabilities needed to issue a Basic Printing Profile simple push and uses a Basic Printing Profile printer as its server.*

Comparison of Basic Imaging Profile Image Push and Basic Printing Profile simple push

First it should be pointed out that the two print mechanisms use different OBEX sessions (also called channels). In the Basic Printing Profile, simple push operations are carried out in a job channel session that is initiated by an OBEX Connect request with the DirectPrinting UUID (see the Bluetooth Assigned Numbers Document for the value) in the Target header. In the Basic Imaging Profile, an Image Push operation is initiated by sending an OBEX Connect request with the Basic Imaging Image Push UUID (see the Basic Imaging Profile specification for the value) in the Target header.

In both profiles, the printer can describe its capabilities to the device performing the push operation. The methods used by the two profiles are slightly different. The Basic Imaging Profile represents this information as an XML string called the Imaging Capabilities object, which is retrieved via the GetCapabilities function. The Basic Printing Profile represents this information in a SOAP encoded GetPrinterAttributes response message.

The Basic Imaging Profile's GetCapabilities request is formatted as follows:

Fields	Opcode	Get
	Packet Length	Length of the packet
Headers	ConnectionID	Connection identifier
	Type	x-bt/img-capabilities

The GetCapabilities response is formatted as follows:

Fields	Response Code	Success or error code
	Packet Length	Length of the packet
Headers	Body/EndOfBody	Imaging Capabilities object

The Basic Printing Profile's GetPrinterAttributes request is formatted as follows:

Fields	Opcode	Get
	Packet Length	Length of the packet
Headers	ConnectionID	Connection identifier
	Type	x-obex/bt-SOAP
	Body header	GetPrinterAttributes SOAP request

The GetPrinterAttributes response command is formatted as follows:

Fields	Opcode	Get
	Packet Length	Length of the packet
Headers	Body header	GetPrinterAttributes SOAP response

The push commands used to send imaging data across are also slightly different, as detailed in the following tables (the differences are highlighted in grey):

In the Basic Imaging Profile, the PutImage request is formatted as follows:

Fields	Opcode	Put
	Packet Length	Length of the packet
Headers	ConnectionID	Connection identifier
	Type	x-bt/img-img
	Name	Image name

	Img-Description	Image descriptor
	Body/EndOfBody	Image file

The PutImage response is formatted as follows:

Fields	Response Code	Success or Partial Content or error code
	Packet Length	Length of the packet
Headers	Img-Handle	Image handle

In the Basic Printing Profile, the equivalent operation is an OBEX Put operation formatted as follows:

Fields	Opcode	Put
	Packet Length	Length of the packet
Headers	ConnectionID	Connection identifier
	Type	MIME media type of object
	Name	Image name
	Body/EndOfBody	Image file

And the associated response is:

Fields	Response Code	Success or Forbidden
	Packet Length	Length of the packet

Realisation of Scenario 1

In this scenario the printer assumes the burden of supporting a second profile.

- The printer must expose the following Service Discovery record:

Service class ID list				See [14]	M	
Service class #0		UUID	Imaging Responder		M	
Protocol descriptor list				See [14]	M	
Protocol ID #0		UUID	L2CAP		M	
Protocol ID #1		UUID	RFCOMM		M	
Param #0	Channel number	Uint8	Varies		M	
Protocol ID #2		UUID	OBEX		M	
Service name	Displayable text name	String	Service-provider defined ("Printing" for instance)	See [14]	O	"Imaging"
Bluetooth profile				See [14]	M	
Profile ID #0	Supported profile	UUID	Imaging			Imaging
Param #0	Profile version	Unit16	0x0100	See [14]		0x0100
Supported capabilities	Imaging capabilities	Uint8	0x80	See [14]	M	0x00
Supported features	Imaging features flags	Uint16	0x2000	See [14]	M	
Supported functions	Imaging functions flags	Uint32	0x40000000	See [14]	M	
Total imaging data capacity	Maximum memory available for image storage	Uint64	Memory in bytes	See [14]	M	

- The printer must accept an OBEX Connect request with the UUID for the Basic Imaging Image Push service in the Target header.

- The printer must handle a GetCapabilities request.

  - The printer must recognise the value "x-bt/img-capabilities" in the Type header as the indication that a request comes from a Basic Imaging Profile device. Note that there isn't a Body header in this request.

  - The printer must respond with an Imaging Capabilities object in the Body header of the response message (instead of a SOAP response). It is recommended that the printer use the following minimal version of the Imaging Capabilities object:

```
<DOCTYPE! Imaging-capabilities [

<!ELEMENT imaging-capabilities ( image-formats*  ) >
<!ATTLIST imaging-capabilities
   version CDATA #FIXED "1.0" >

<!ELEMENT image-formats EMPTY>
<!ATTLIST  formats
   encoding CDATA #REQUIRED
   pixel CDATA #IMPLIED
   maxsize CDATA #IMPLIED>

>
```

* Note that this step should not imply any modification to the XML encoder implementation used by the printer. It is sufficient to have the above XML string pre-coded statically.

- The printer must handle a PutImage request.
    - The printer must recognise the value “x-bt/img-img” in the Type header as the indication that the OBEX Put request comes from a Basic Imaging Profile device that is pushing an image.
    - The printer must either process or ignore the Img-Description header. In no case should it reject the Put request due to the presence of the Img-Description header. It is recommended, but not mandatory, that the printer be capable of reading the Image Descriptor to retrieve at least the encoding of the incoming image file.
    - The printer must return an Image Handle in the response message. It is allowable to use a fixed value like “0000000”.

<u>Realisation of Scenario 2</u>

In this scenario the camera assumes the burden of supporting a second profile.

- The camera must issue an OBEX Connect request with the DirectPrinting UUID in the Target header.
- The camera must issue a simple push command:
    - The Type header in the PutImage command must be modified so that it carries the MIME media type of the image instead of the Basic Imaging Profile’s usual x-bt/img-img.
    - The Img-Description header must not be used.
    - The camera must accept a PutImage response without an Img-Handle header.

Note that in Scenario 2, the camera can’t support the Basic Printing Profile’s GetPrinterAttributes without also supporting the CreateJob command, which is part of the Basic Printing Profile’s job-based transfer model. The camera can, however, obtain significant information (including the document formats supported by the printer) from the printer's Service Discovery record.

Q8. How is security handled in the Basic Imaging Profile?

The imaging working group did not introduce any security mechanisms beyond those already available in the Bluetooth 1.1 specification; in particular the standard baseband and OBEX security mechanisms (described in the Generic Access Profile and Generic Object Exchange Profile, respectively) apply to the Basic Imaging Profile.

Q9. Is there a concept of content protection in the Basic Imaging Profile?

The Basic Imaging Profile does not define a content or copyright protection mechanism, although such a mechanism could be implemented on top of the profile's image exchange features.

Q10. What is the role of the ServiceID SDP attribute? How is it used in the profile?

The ServiceID attribute, as defined in the Service Discovery Protocol specification, is a UUID that uniquely identifies the service instance described by a service record. The need to uniquely identify a service instance emerges when a number of applications that support the same profile coexist on one device.

More precisely, some of the Basic Imaging Profile's features make use of two OBEX sessions running in parallel with reversed client and server roles: the client in the first OBEX session is the server in the second. The second session is opened as result of a request issued within the first session and is preceded by the first session's server looking for channel information in the relevant service record on the client device. Assume for a moment that there is more than one application on the client device that can handle such a second session: this creates a problem for the server as it tries to locate the correct service record – that is, the service record associated with the application that requested the second session. The solution to this problem is a unique service record identifier, which is used as follows: when the first session's client issues a request for a second session, it sends the unique identifier of the service record the server device should consult to retrieve channel information. The server device runs a Service Discovery Protocol search for the service record with that unique identifier in its ServiceID attribute, retrieves the channel information, and establishes the second session.

Note that the UUID can be freely chosen; there isn't a specific range of UUID associated with the ServiceID attribute.

Q11. How do I discover/identify a device supporting the Basic Imaging Profile?

The Bluetooth baseband's Class of Device/Service (CoD) field contains information that can be used to screen devices for possible Basic Imaging Profile support. In particular, the Major Service Class portion of the CoD includes Rendering, Capturing, and Object Transfer bits. A digital still camera that supports the Basic Imaging Profile should have the Capturing and Object Transfer bits set, while a printer that supports the profile should have the Rendering and Object Transfer bits set. The Major Class portion of the CoD includes an Imaging value with associated Minor Device Class values of Display, Camera, Scanner, and Printer (note that these Minor Device Class values can be bitwise OR'ed together). A digital still camera will probably have its Major Class set to Imaging and its Minor Device Class set to Camera, while a printer will probably have its Major Class set to Imaging and its Minor Device Class set to Printer.

Testing the baseband CoD field is not guaranteed to locate all imaging devices, however. A Bluetooth-equipped PC with a (cabled) printer, for example, could act as a proxy for the Advanced Image Printing feature of the Basic Imaging Profile, and its CoD bits would probably fail the screening test described above; at the other extreme, a printer that supports only the Basic Printing Profile would pass the screening test but obviously wouldn't support the Basic Imaging Profile. The only sure method to test a device for Basic Imaging Profile support is to perform a Service Discovery Protocol inquiry, searching for the Imaging UUID in the Profile Descriptor List (note that the Basic Imaging Profile mandates the presence of the Profile Descriptor List attribute, which is often optional in other profiles).

Q12. Why is the Image Push feature divided into four sub features in the Imaging Responder service record?

The Imaging Responder service record includes a Supported Features attribute that details the level of support for the various features provided by the Basic Imaging Profile. While the profile defines six features (Image Push, Image Pull, Advanced Image Printing, Automatic Archive, Remote Camera, and Remote Display), the Supported Features attribute subdivides the Image Push feature into four sub-features: Image Push, Image Push-Display, Image Push-Print, and Image Push-Store. This subdivision was introduced to allow a server to advertise the outcome of an image push operation. This notion is best illustrated by a concrete example: imagine a multi-purpose device like a PC that has the ability to do anything with an image, and a number of Basic Imaging Profile applications running on that device that have different functions: Application1 is a photo album application, Application2 is print spooler, Application3 is a file manager, and Application4 is a multi-purpose application that can display, print, or store images.

A client device that performs a Service Discovery Protocol inquiry on this multi-purpose server device would find four service records, one for each application. By consulting the Supported Features attribute of each service record, the client can make an educated guess about which service record/application it should target for an image push operation based on the desired outcome: Application1 would advertise the Image Push-Display and Image Push-Store features, Application2 would advertise Image Push-Print, Application3 would advertise Image Push-Store, and Application4 would advertise Image-Push.

Q13. How should I use the friendly-name attribute in the Image Properties object?

The Basic Imaging Profile uses image handles to identify images to guarantee that images are uniquely indexed. Image handles have the following drawbacks, however:

1. Image handles are not user friendly.

2. Image handle uniqueness is guaranteed only for the duration of a single OBEX session. This limitation can have confusing effects for users. If, for example, a user pushes an image to a destination device in one OBEX session, then attempts to pull it in a different OBEX session, there's no guarantee the image handle returned after the push operation in the first session will successfully pull the same image in the later session – the image handle may have changed.

To alleviate this drawback, the profile defines a “friendly-name” attribute in the Image Properties object that associates a human readable name with an image. This friendly name can be displayed to users at the man-machine interface level while image handles are used to identify and exchange images at the profile protocol level. Friendly names can also be used to associate images across OBEX sessions, although friendly names are not guaranteed to be unique.

Q14. How is the name attribute used to identify image attachments?

While the Basic Imaging Profile uses image handles to uniquely identify images, it uses string names to identify attachments associated with images. These names need to be unique so attachments can be uniquely identified. It is anticipated that in some implementations, associating a unique name with an attachment may be difficult; using the full folder/directory path as part of the name should be a workable solution in most cases.

Q15. How is the Name header used in a PutImage operation?

The Name header included in the PutImage request is designed as a mechanism for the Initiator to propose to the Responder a human readable name for the image that is being pushed. The name may or may not be related to the location and designation of the image in the Initiator’s file system. The name may also be empty (0x01 as HI, and 0x0003 as length value) in the case of very low end Initiators. It is important to note that the name proposed by the Initiator does not necessarily comply with the naming conventions imposed by the Responder’s file system. If the name proposed by the Initiator is unusable by the Responder, the Responder will have to generate its own human readable name; the Responder can completely ignore the name proposed by the Initiator or it can modify the proposed name to make it useable.

Q16. What is the recommended usage of the OBEX Abort command?

The interpretation of the OBEX Abort command is somewhat controversial. Most existing OBEX implementations handle the Abort command in exactly the same way as other OBEX commands. Some interpretations of the OBEX specification suggest that the Abort command differs from other commands in that it can be issued by a client and processed by a server at any time, regardless of the state of the current operation; in particular, a client can send an Abort command request without waiting for a response from the server to a previous request, violating the ping-pong nature of the OBEX protocol.

The Basic Imaging Profile adopts the simpler interpretation – the Abort command is only issued after receiving a response to the previous request. This is sufficient for the Basic Imaging Profile and avoids placing additional requirements on existing OBEX implementations. The one case where this approach can be problematic is when a client issues an OBEX Get request associated with a GetImagesList request and the server takes a long time processing that request – the client may wish to abort the Get request before it receives the first response from the server. The best implementation strategy in this case is to use a timer in the client that results in closing the OBEX session if the server takes too much time.

Q17. My implementation cannot handle big data files. I need to negotiate the size of image files in addition to format and pixel size. Is this possible with the Basic Imaging Profile?

Full blown image file size negotiation is not supported by the Basic Imaging Profile. Such negotiation would require that a server implementation:

1. Know the size of an image file for each supported pixel size, encoding, and format combination. This is difficult to know without actually performing image transformation and encoding for each combination, which is computationally very expensive.

2. Adjust compression/encoding parameters while performing format conversions requested by a client. Most existing imaging devices operate with fixed parameters tailored to that device.

While full file size negotiation is not supported, the Basic Imaging Profile does include mechanisms that can be used to place an upper bound on image file sizes. A full description of these mechanisms requires distinguishing between image push and image pull scenarios.

<u>Image file size handling in Image Push operations</u>

The issue here is to protect the receiving (server) device from large image files it cannot process. The recommended strategy is as follows: the receiving device should set the “maxsize” attribute of its Imaging Capabilities object to the maximum file size it is prepared to handle. When the device performing the image push operation retrieves the Imaging Capabilities object, it should read the “maxsize” attribute and refrain from pushing image files that are larger than the value indicated.

There is no guarantee, however, that the device performing the image push operation will retrieve the Imaging Capabilities object or parse and use the “maxsize” attribute. Therefore the receiving device should also interpret the Image Descriptor that always accompanies an image push operation, in particular its “size” attribute. By reading the “size” attribute, the receiving device can immediately determine whether or not it can accommodate the image file it’s about to receive. If the size is too large, the receiving device can terminate the operation, preferably with the OBEX “precondition failed” response code if it’s supported, otherwise “bad request”. The device performing the image push can subsequently retrieve the Imaging Capabilities object of the receiving device and check the “maxsize” attribute, try to push the image again in a different format, or give up.

<u>Image file size handling in Image Pull operations</u>

In an Image Pull scenario, the device that is providing the images is strongly encouraged to supply the “maxsize” attribute in the Image Properties object associated with each image file. In this context, the “maxsize” attribute gives an indication of what file size to expect. As previously explained, it is very difficult to provide an exact size value without actually performing the transformation/encoding first, so an approximate value is acceptable, provided that the approximation is an upper bound on the file size.

Before pulling an image, the retrieving device reads the Image Properties object from the server to make its choice of encoding and pixel size. If the retrieving device intends to pull an image in a variant encoding, it’s encouraged to take into account the “maxsize” attribute provided by the server. If the server does not provide a “maxsize” attribute in the Image Properties object, the retrieving device should take note of the OBEX Length header in the GetImage response, which indicates the total size in bytes of the image carried in the Body header; this Length header is always available in the first response packet, even for multi-packet responses. The retrieving device can therefore use this information to issue an OBEX Abort if the image file is too large.

Q18. Should the Imaging Thumbnail format always be listed as available format in an Image-Properties object?

Yes, the Image-Properties object must always include the Imaging Thumbnail format as an available format. This can be formulated in several ways, however. The Imaging Thumbnail format could be the native version of an image, in which case it appears in the “native” XML element with “JPEG” as the “encoding” attribute and “160*120” as the “pixel” attribute. If Imaging Thumbnail is not the native format, it must appear in one of the “variant” elements, where it could be described as:

1. One “variant” element with a fixed “pixel” attribute (<variant encoding=”JPEG” pixel=”160*120”), or

2. Part of a “variant” element with a ranged “pixel” attribute (for instance, <variant encoding=”JPEG” pixel=”80*60-640*480”), or

3. A redundant “variant”, meaning that although Imaging Thumbnail support be deduced from a ranged “variant” as indicated in 2), it can also be repeated in the form of a variant with a fixed “pixel” attribute as in 1).

Q19. Can a very simple Image Pull Initiator skip the GetImageProperties function?

Yes, it is possible to skip the GetImageProperties function. There are several scenarios where this can be considered:

1. A device is only interested in retrieving images in the mandatory Imaging Thumbnail format. Given that all the images on a Responder must be available in Imaging Thumbnail format, there is no point in retrieving the Image-Properties object to verify that Imaging Thumbnail is one of the supported formats.

2. The number of formats that a device can utilize is very limited. In that case, it may be faster to attempt to retrieve images in formats the device understands (a trial and error approach) rather than retrieve the Image-Properties object, check the supported formats, choose one format, and retrieve an image in that format. Obviously, a trial and error approach presents challenges at the user interface level.

3. A device uses the Image-Handles descriptor in the GetImagesList function to filter the list of available images by the “encoding” (and possibly by the “pixel”) attribute, then uses the information in the Images-Listing object returned by the

GetImagesList function to retrieve images. Note that this is subject to the Responder's ability to process the "encoding" and "pixel" filtering parameters, which are optional to support (whether or not a Responder supports these filtering parameters is indicated in the Responder's Imaging-Capabilities object). If the Responder supports filtering of the Images-Listing object by the "encoding" (and possibly by the "pixel") attribute and the device is capable of ordering the result of such filtering, the device may want to skip the GetImageProperties function. The device should make sure that the "encoding" and "pixel" attributes in the Image-Descriptor supplied to a subsequent GetImage function match those used in the Image-Handles descriptor in the GetImagesList request. Note in particular that if the "pixel" attribute takes the form of a range, the Responder will be free to return the image with whatever pixel size it chooses as long as it complies with that range.

Q20. What is an empty Image Descriptor?

When requesting a native image using the GetImage function, the Image Descriptor must be empty. The meaning of empty here is this: the Img-Description header shall contain only the header ID for Img-Description (0x71) and the length value 0x0003 (two byte big-endian unsigned integer indicating that the length of the header is three bytes).

Q21. Can a Remote Display Responder accommodate a Remote Display Initiator that is not supporting the 'Select' command ?

Provided the PutImage function is supported by the Initiator device, the answer to this question is yes. The PutImage function is mandatory in the Remote Display feature. This feature can therefore not be used to implement a very simple remote controller device that would only pilot the Responder. A Remote Display Initiator must have the ability to actually send pictures to the Responder.

When the Initiator is not supporting the Select command and the Responder is initially not displaying any image, although the profile gives total liberty to the Responder implementation to choose how to handle a Next or a Select command, it is recommended that upon reception of a Next command the Responder displays the first image in the list that it maintains of the images received from the Initiator, and upon reception of a Previous command the Responder displays the last image in this same list.

It can be expected that most implementations of a Remote Display Responder will conserve the order of the images as received from the Remote Display Initiator (i.e. the list of images on the Responder follows the same order as the order under which the images were transferred from the Initiator to the Responder). Implementers of a Remote Display Initiator should nevertheless be aware that, for the sake of flexibility, the conservation of the order of the images is not imposed by the profile and that therefore, depending on the usage cases, there might be Responders that choose to rearrange the images received from the Initiator in a different order.