X IMAGE EXTENSION PROTOCOL

Revision/Update Information: V2.1

Digital Equipment Corporation

Contents

PREFACE v

CHAPTER 1 INTRODUCTION 1–1

1.1 SCOPE AND PURPOSE 1–1

1.2 TERMINOLOGY 1–1

1.3 CONCEPTS RELATIONSHIP 1–1

CHAPTER 2 PROTOCOL MESSAGE FORMATS 2–1

2.1 REQUEST PROTOCOL MESSAGE FORMAT 2–1

2.2 REPLY PROTOCOL MESSAGE FORMAT 2–2

2.3 ERROR PROTOCOL MESSAGE FORMAT 2–2

2.4 EVENT PROTOCOL MESSAGE FORMAT 2–3

2.5 DATA ENCODING 2–4

CHAPTER 3 PROTOCOL SPECIFICATION SYNTAX 3–1

3.1 GENERAL SYNTAX 3–1

3.2 REQUEST SYNTAX 3–1

3.3 ERROR SYNTAX 3–2

3.4 EVENT SYNTAX 3–3

CHAPTER 4 PROTOCOL PARAMETER TYPES 4–1

4.1 CORE TYPES 4–1

4.2 EXTENSION TYPES 4–1

CHAPTER 5 PROTOCOL ERROR CODES 5–1

5.1 CORE ERRORS 5–1

5.2 EXTENSION ERRORS 5–2

CHAPTER 6 SERVER DATA MODEL 6–1

6.1 CONSTRAINED VS UNCONSTRAINED 6–1

6.2 MONO VS POLY CHROMATIC 6–3

6.3 DATA FLOW PIPELINING 6–3

Contents

CHAPTER 7 REQUESTS AND REPLIES 7–1

7.1 SESSION SERVICES 7–2
7.1.1 Initialize Session 7–2
7.1.1.1 Per Server information • 7–3
7.1.1.2 Per Screen Information • 7–3
7.1.1.3 Relationship to Core Session Information • 7–3
7.1.2 Set Operational Defaults 7–4
7.1.3 Query Operational Defaults 7–5
7.1.4 Terminate Session 7–6

7.2 RESOURCE SERVICES 7–7
7.2.1 Resource Creation 7–7
7.2.1.1 Image Object Resource • 7–8
7.2.1.2 Domain Control Context Resource • 7–9
7.2.1.3 Process Control Context Creation • 7–11
7.2.2 Resource Copy 7–12
7.2.3 Resource Delete 7–12
7.2.4 Resource Query 7–13
7.2.4.1 Image Object Query • 7–13
7.2.4.2 Domain Control Context Query • 7–14
7.2.4.3 Process Control Context Query • 7–15
7.2.5 Import/Export Services 7–15
7.2.5.1 Import Image • 7–16
7.2.5.2 Export Image • 7–17

7.3 TRANSPORT SERVICES 7–18
7.3.1 Overview of Data Transport 7–18
7.3.1.1 Transport Process • 7–18
7.3.1.2 Transport Modes • 7–19
7.3.1.3 Transport Data Formats • 7–19
7.3.1.4 Transport Granularity & Atomicity • 7–21
7.3.1.5 Synchronization • 7–22
7.3.2 Set Transport Context 7–22
7.3.3 Tile Data Transport 7–23
7.3.3.1 Client–>Server • 7–23
7.3.3.2 Server–>Client • 7–25
7.3.4 Stream Data Transport 7–26
7.3.4.1 Client–>Server • 7–27
7.3.4.2 Server–>Client • 7–28
7.3.5 Abort Transport 7–30

7.4 PROCESS SERVICES 7–31
7.4.1 Point Processing 7–33
7.4.2 Area Processing 7–35
7.4.3 Ensemble Processing 7–37
7.4.3.1 Arithmetic • 7–37
7.4.3.2 Relational • 7–38
7.4.3.3 Logical • 7–39
7.4.3.4 Mathematical • 7–40

7.4.4 Geometric **7–42**
7.4.4.1 Crop Image • 7–42
7.4.4.2 Translate Image • 7–43
7.4.4.3 Scale Image • 7–44
7.4.4.4 Rotate Image • 7–45
7.4.4.5 Mirror Image • 7–46
7.4.5 Radiometric **7–47**
7.4.5.1 Constrain • 7–47
7.4.5.2 Histogram Matching • 7–48
7.4.5.3 Separate Spectral Components • 7–49
7.4.5.4 Combine Spectral Components • 7–49
7.4.5.5 Dither • 7–50
7.4.5.6 Luminance • 7–51

7.5 STATISTICS SERVICES **7–52**
7.5.1 Component Distribution **7–52**
7.5.2 Point Statistics **7–53**
7.5.3 Area Statistics **7–54**

7.6 PROCESS CONTROL SERVICES **7–55**
7.6.1 Adding Segments **7–55**
7.6.2 Executing a Pipe **7–56**
7.6.3 Terminating a Pipe **7–57**

7.7 DISPLAY SERVICES **7–58**

7.8 EVENT SERVICES **7–59**
7.8.1 Select Events **7–59**
7.8.2 Query Events **7–59**

CHAPTER 8 EVENTS **8–1**

8.1 COMPUTATION COMPLETE **8–1**

8.2 PIPELINE COMPLETE **8–1**

8.3 SEQUENCE COMPLETE **8–2**

CHAPTER 9 APPENDICES **9–1**

9.1 XIE PROTOCOL SUMMARY **9–1**

9.2 RECOMMENDED SERVER RESOURCE CONTENTS **9–3**
9.2.1 IOB **9–3**
9.2.2 IDC **9–4**
9.2.3 IPC **9–5**

9.3 IMAGE DATA ATTRIBUTES & ORGANIZATION **9–6**
9.3.1 Primitive Attributes **9–7**
9.3.2 Organization & Encoding Attributes **9–7**
9.3.2.1 Encoding Attributes • 9–8
9.3.2.2 Component Space Attributes • 9–8
9.3.3 Presentation Attributes **9–10**

9.4 CONCEPT AND VERSION CHANGES **9–11**
9.4.1 Conceptual Changes **9–11**

Contents

9.4.2 V1.0 - V2.0 Protocol Changes 9–12

9.5 REFERENCES 9–14

Preface

The core X architecture is a major step in the definition of open, interactive applications display services. It has, in essence, pried apart the tightly coupled, private interconnect which has existed in the past between application and display subsystem by specifying a structured set of display services.

The core X protocol recognizes the need for minimally sufficient capabilities, and, as such, leaves the specification of robust application domain specific display services to extension packages. These extension packages relate to the core X protocol by way of an extension mechanism specified in the core X protocol.

This document contains the X wire protocol specification for the proposed X Image Extension Package. It is assumed that the reader is familiar with the core X architecture, the X11 wire protocol, and the proposed X Image Extension Concepts.

Related Documents

- The X Window System (Scheifler & Gettys)

  Provides an overview of the X windowing system and architecture.
- X11 Window System Protocol

  Describes the core X11 protocol and conventions used in specifying extension packages.
- X Extension Package Guidelines

  Sets forth guidelines for extension packages.
- X Image Extension Concepts

  Describes the architectural concepts behind this protocol specification. The X Image Extension Concepts document is one several companion documents to this protocol specification.
- X Image Extension Server Guidelines

  Provides an overview of the design of an image extension server. The X Image Extension Server Guidelines document is one of several companion documents to this protocol specification.

Participants

- Author:
  - Joseph Mauro
  - Digital Equipment Corporation
  - MS: MK2-2/K5
  - Continental Blvd.
  - Merrimack, New Hampshire 03057

  - 603-884-0528
  - MAURO@OPTIC.DEC.COM

- Contributors:
  - Olliver Jones (Apollo)
  - Barry Medoff (SUN)
  - John Weber (Digital ISG)
  - Rich Piccolo (Digital ISG)
  - Robert Shelley (Digital ISG)

- Invited Reviewers:
  - Rebeca Hinkle (Apollo)

Revision History

rev v1.0 August 1, 1988 first protocol draft

rev v2.0 April 12, 1989 major changes based upon consortia review

rev v2.1 November 1, 1989 changes based upon FT1 implementation

Organization

This document is organized into the following chapters and appendices:

- Chapter 1, Introduction, explains the purpose of the X Image Extension
- Chapter 2, Protocol Message Formats, defines structure of protocol messages
- Chapter 3, Protocol Specification Syntax, defines protocol function syntax
- Chapter 4, Protocol Parameter Types, defines types for protocol parameters
- Chapter 5, Protocol Error Codes, defines possible protocol errors
- Chapter 6, Server Data Model, server data organization and operator issues
- Chapter 7, Requests and Replies, describes the main protocol functions
- Chapter 8, Events, describes the various extension related events
- Appendices:
  - XIE Protocol Summary
  - Recommended Server Resource Contents
  - Image Data Attributes & Organization
  - Concept and Version Changes
  - References

Chapter 1

INTRODUCTION

1.1 Scope and Purpose

This document specifies the X wire protocol for the X Image Extension (XIE). This extension protocol is deemed to be sufficient to support implementation of the image concepts found in the X Image Extension Concepts document. This specification defines the syntax, structure, and semantics of X wire image extension protocol elements.

This specification does not contain any background material useful in understanding the imaging services concepts that the extension protocol enables. This background material can be found in the extension concepts document.

This specification does not contain any language function bindings. These will be left to a follow on document which will specify the language bindings for the X Image Extension to Xlib (XIElib).

1.2 Terminology

This specification makes use of two classes of terminology: descriptive image terminology, and parametric terminology. The descriptive image terminology is defined in a glossary in the X Image Extension Concepts document and is used throughout this protocol specification. Readers should consider reviewing this material before reading this document.

The parametric terminology defines the parameter types recognized in the core X11 protocol and in the image extension protocol. Image extension protocol types are defined in this document.

1.3 Concepts Relationship

The X Image Extension Concepts document describes the reasoning behind an X image extension and lays the foundation for this protocol specification. Conceptual shifts since the concepts document was drafted, and major protocol differences between versions 1 and 2 of this specification are briefly described in the "concept and version changes" appendix. While there have been protocol changes between version 2.0 and 2.1 (deriving from the field test implementation), they do not reflect major conceptual shifts.

Chapter 2

PROTOCOL MESSAGE FORMATS

XIEP follows the same operational model as set forth by the X core protocol specification regarding the use of requests, replies, errors, and events, and the overall structure of these protocol objects. This section defines the overall structure of these protocol objects for the image extension protocol.

2.1 Request Protocol Message Format

All image extension requests comply with the overall extension request format as specified in the X11 core protocol specification request format section. This section reiterates that format as applied to the image extension.

An image extension request consists of 4 bytes of header followed by zero or more bytes of data. The four byte header complies exactly with the core protocol request format (1 byte major opcode - 1 byte minor opcode - 2 bytes total length field). Unused bytes in a request are not required to be zero.

```
 REQUEST FORMAT
 --------------

BYTE  CONTENTS

0  major opcode
1  minor opcode - extension function code
2  length field (LSB)
3  length field (MSB)
4  start of data - (request specific)
 .
 .
 .
```

The major opcode field contains the opcode signaling a request for the image extension package (between 128 and 255). The value is determined by the core X server when the extension is loaded. All discussion of minor opcodes in this document refer to image extension requests (op codes) where the major opcode is that for the image extension.

The minor opcode field contains the image function code. Individual image extension function codes will be found in the image extension server guide.

The length field defines the total length of the request including header and data in units of 4 bytes. The length field must equal the minimum length required to contain the request or an error is generated. The particular contents of the data field portion of a request are defined on a request by request basis (see requests section).

Every request is implicitly assigned a sequence number, starting with one, used in replies, errors, and events. {This sequence number is the one assigned by the core server - the extension does not assign any sequence number, but rather relies upon the core assigned sequence number}

2.2 Reply Protocol Message Format

The reply format follows that specified in the core protocol (below). The particular contents of the data portion of a reply are defined on a request by request basis (see requests section).

Every reply contains a 32-bit length field expressed in units of 4 bytes. Every reply consists of 32 bytes, followed by zero or more additional bytes of data, as specified in the length field. Unused bytes within a reply are not guaranteed to be zero. Every reply also contains the least significant 16 bits of the sequence number of the corresponding request.

```
REPLY FORMAT
------------

        BYTE            CONTENTS

        0               reply type msg (value = 1)
        1               unused
        2               request sequence number
        3               request sequence number
        4               length field
        5               length field
        6               length field
        7               length field
        8               start of data - (request specific)
              .
              .
              .
```

2.3 Error Protocol Message Format

The error format follows that specified in the core protocol (below).

Error reports are 32 bytes long. Unused bytes within an error report are not guaranteed to be zero. Every error report includes the 8-bit major opcode, the 8 bit minor opcode (extension function), and the 8 bit error code.

```
 ERROR FORMAT
 ------------

BYTE  CONTENTS
```

```
 0  error type msg (value = 0)
 1  error code
 2  request sequence number
 3  request sequence number
 4  unused
 5  unused
 6  unused
 7  unused
 8  minor opcode - extension function code
 9  minor opcode - extension function code
10  major opcode
```

The major and minor (function code) opcodes, are those of the request which generated the error. The request sequence number is the least significant 16 bits of the number implicitly assigned to the request which generated the error.

For the following errors (see Section n), the failing resource id is also returned:

- Core Protocol Errors
  - Colormap
  - Cursor
  - Drawable
  - Font
  - GContext
  - IDChoice
  - Pixmap
  - Window

For Atom errors, the failing atom is returned. For Value errors, the failing value is returned.

2.4 Event Protocol Message Format

The event format follows that specified in the core protocol (below).

Events are 32 bytes long. Unused bytes within an event are not guaranteed to be zero. Every event contains an 8-bit type code. The most significant bit in this code is set if the event was generated from a SendEvent request.

The image extension related events use event codes assigned by the core server at extension load time. Every core event (with the exception of KeymapNotify) also contains the least significant 16 bits of the sequence number of the last request issued by the client that was (or is currently being) processed by the server. Image extension events also include the request sequence number.

```
 EVENT FORMAT
 ------------

BYTE  CONTENTS

0  event type code (value = )
1  request sequence number
2  request sequence number
```

2.5 Data Encoding

Except where explicitly noted in the protocol, all 16 bit and 32 bit quantities exchanged between client and server must be transmitted with the byte order established at CORE protocol connection time. (see core protocol connection setup - X11 spec.). In particular...

```
"The client must send an initial byte of data to identify the byte order to be
employed. The value of the byte must be octal 102 or 154. The value 102 (ASCII
upper case B) means values are transmitted most significant byte first, and
value 154 (ASCII lower case l) means values are transmitted least significant
byte first. Except where explicitly noted in the protocol, all 16-bit and
32-bit quantities sent by the client must be transmitted with this byte order,
and all 16-bit and 32-bit quantities returned by the server will be transmitted
with this byte order."
```

Chapter 3

PROTOCOL SPECIFICATION SYNTAX

This section presents syntactic conventions adhered to throughout the specification. All protocol functions (requests, replies, errors, events) are specified according to this syntax.

3.1 General Syntax

In general, the extension package follows the X11 protocol syntax conventions.

The syntax {...} encloses a set of alternatives.

The syntax [...] encloses a set of structure components.

TYPE's are in upper case, and Alternative Values are capitalized.

3.2 Request Syntax

Requests (and the associated replies) are described in this document in a uniform fashion. There are five major sections to a request description:

1. request "procedure declaration"
2. errors
3. events
4. overall description

The first three are specified in a brief BNF like format. The overall description is in English prose and follows the first three.

Each request description begins with a "procedure declaration" like definition consisting of the request name, argument names and types, and optional reply information. All image extension requests have a common name prefix (XIEP_). The arguments constitute the optional data described in the request protocols on a protocol function by protocol function basis. Note that the order of arguments complies with their actual order in the wire protocol. Note also that the protocol function names and types are abstract (the XieLib language bindings convert these abstract names into concrete function names an parameter types).

The symbol "=>" demarcates the start of reply information. If the request has no reply, the word "none" will follow the symbol "=>".

Following the "procedure declaration" information is a list of errors which the protocol function may signal. If there are no possible error conditions, the list consists of the word "none".

Following the error information is a list of events which the protocol function may trigger. If there are no possible events which maybe triggered, the list consists of the word "none". Note that the list contains only those events which may be explicitly triggered, and that there may be unforeseen events triggered by a protocol function.

Finally, each request section contains a descriptive component which includes overall purpose, parameters , and operation.

```
RequestName
  arg1:  TYPE1
       .
       .
       .
  argn:  TYPEN
 =>
  result1: TYPE1
       .
       .
       .
  resultn: TYPEN

Errors: kind1, ..., kindN

Events: kind1, ... ,kindN

Description:
  - purpose
  - parameters
  - operation
           - errors
```

3.3 Error Syntax

Errors are described in a uniform fashion. Each error description includes the error name, error type code, and optional information returned with the error. Finally, each error section contains a descriptive component which addresses causes.

```
ErrorName error_type_code
  value1: TYPE1
    .
    .
    .
  valuen: TYPEN
Description:
  - causes
```

3.4 Event Syntax

Events are described in a uniform fashion. Each event description includes the event name, event type code, and optional information returned with the event. Finally, each event section contains a descriptive component which includes overall purpose, causes, and operational semantics.

```
EventName event_type_code

  value1: TYPE1
    .
    .
    .
  valuen: TYPEN

Description:

  – overall purpose
  – causes
  – operational semantics
```

Chapter 4

PROTOCOL PARAMETER TYPES

Parametric types represent the core and extension protocol recognized types for the various request parameters. The image extension protocol makes use of both core protocol parametric types and image extension specific protocol parametric types.

4.1 Core Types

The following types are defined in the X11 core protocol specification. Refer to the X11 core protocol specification for their definition.

ARC	ATOM	BOOL	BITGRAVITY
BITMASK	BUTMASK	BUTTON	CARD8
CARD16	CARD32	CHAR2B	COLORMAP
CURSOR	DEVICE	DEVICEEVENT	DRAWABLE
EVENT	FONT	GCONTEXT	HOST
INT8	INT16	INT32	KEYBUTMASK
KEYCODE	KEYMASK	LISTofFOO	LISTofVALUE
Or Types	PIXMAP	POINT	POINTEREVENT
RECTANGLE	STRING8	STRING16	TIMESTAMP
VALUE	VISUALID	WINDOW	WINGRAVITY

Note that XIE redefines the use of LISTofFOO such that the list is preceded by the number of entries in the list. This applies in all cases (whether FOO is an XIE specific or core type). The number of entries field is CARD8 unless noted otherwise in a protocol function specification.

4.2 Extension Types

The types presented in this section are specific to the X Image Extension Protocol, and are considered to be defined types. All X image extension type symbols begin with the prefix XIT_. All X image extension protocol parameters are defined as being either one of these extension types, or one of the core protocol types. Note that these types are abstract in nature and are bound to concrete types and structures by the XieLib language bindings.

The following sections describe the particulars of each of the image extension types.

XIT_ACCELERATOR_VECTOR

XIT_ACCELERATOR_VECTOR is used to provide the client information about which protocol functions can be executed on ancillary processors (one other than the extension server host processor). The ancillary processor is typically present for acceleration purposes (thus the name).

Accelerator information is formatted as a bit vector identical in format to that used in server_ capabilities. In particular, there is a bit for every protocol function whose value indicates the state of hardware support for that function on the server. A '0' implies no direct hardware acceleration (ie. the protocol function is implemented through software on the server host processor), while a '1' implies specific accelerator support (ie. a processor other than the server host processor is used to accelerate the function).

XIT_AREA_OP1

XIT_AREA_OP1 defines the valid operations for the process services area operator number 1. They include:

```
XIT_AREA_OP1
            {
             ADDITION,
             MULTIPLICATION
            }
```

XIT_AREA_OP2

XIT_AREA_OP2 defines the valid operations for the process services area operator number 2. They include:

```
XIT_AREA_OP2
            {
             MINIMUM,
             MAXIMUM,
             SUMMATION
            }
```

XIT_ARITH_OP

XIT_ARITH_OP defines the valid operations for the process services arithmetic operator. They include:

```
XIT_ARITH_OP
           {
            ADD_II, ADD_IC,
             SUB_II, SUB_IC, SUB_CI,
             MUL_II, MUL_IC,
             DIV_II, DIV_IC, DIV_CI,
             MIN_II, MIN_IC,
             MAX_II, MAX_IC
            }
```

Note that the suffix II means that the operation is to be performed between two images, while IC means between an image and a constant. Note also that IC, and CI suffices imply order of operands in the operation.

XIT_COMPONENT_SPACE_ORGANIZATION

XIT_COMPONENT_SPACE_ORGANIZATION defines the way in which the various spectral components of an image are organized during wire transport. See the appendix on image data attributes & organization for more details.

```
XIT_COMPONENT_SPACE_ORGANIZATION:

            {
             BAND_INTERLEAVED_BY_PIXEL,
             BAND_INTERLEAVED_BY_PLANE,
             BIT_INTERLEAVED_BY_PLANE
            }
```

XIT_COMPRESSION_SCHEME

XIT_COMPRESSION_SCHEME defines the recognized image compression schemes.

```
XIT_COMPRESSION_SCHEME: {PCM, G42D, DCT}
```

The X Image Extension Protocol makes use of standardized image data specific compression schemes (general purpose information theoretic schemes such as Liv Zempel are thus excluded). While these may not be the most current or efficient, this ensures that incompatibilities due to proprietary or ill defined schemes does not occur.

The relationship between data organization and data encoding (compression) at wire transport, and the applicability of various compression schemes to various classes of digital images under various organizations is discussed in more detail in the section on wire data transport overview, and in appendix C (Image Data Attributes & Organization).

PCM or pulse code modulation implies no compression scheme has been applied to the data. In this case, the image data at wire transport time is organized according to the selected (client-server agreed) format, but no compression scheme has been applied to the data itself. It is included as a "compression scheme" so that it may be transported in stream format (see XIEP_GET_STREAM and XIEP_PUT_STREAM).

G42D stands for the CCITT group 4 two dimensional compression scheme. While this scheme was first developed for bitonal facsimile images, it is applicable to bitonal and continuous tone images when organized properly. The name of this scheme will be used as a marker for the actual standardized algorithm and data tables (refer to CCITT references). In this case, the image data at wire transport time is organized according to the selected (client-server agreed) format, and within this agreed to organization, the data is compressed according to this scheme.

DCT or discrete cosine transform is a technique currently being standardized by ISO and will be included for continuous tone monochromatic, and continuous tone polychromatic compression.

XIT_IDATA

XIT_IDATA is the generic type for all image data which flows across the wire.

XIT_IDC

XIT_IDC is the type for an image domain context identifier. It is defined as a 32 bit identifier. See the notes on resource id formats in the section on connection setup.

XIT_IDC_TYPE

XIT_IDC_TYPE identifies the valid types of image domain contexts (IDC's). They include:

```
XIT_IDC_TYPE:
             {
              CPP - control process plane,
              ROI - region of interest,
              TMP - area operation template
             }
```

XIT_IOB

XIT_IOB is the type for an image object identifier. It is defined as a 32 bit identifier. See the notes on resource id formats in the section on connection setup.

XIT_IPC

XIT_IPC is the type for an image process control context identifier. It is defined as a 32 bit identifier. See the notes on resource id formats in the section on connection setup.

XIT_HSHAPE

XIT_HSHAPE defines the various algorithm shapes that can be requested in histogram matching operations. They include:

```
XIT_HSHAPE
          {
           flat,
           ramp,
           hyperbolic,
           gaussian
          }
```

XIT_LOGICAL_OPS

XIT_LOGICAL_OPS defines the valid operators of the logical process services. They include:

```
XIT_LOGICAL_OPS
                {
                 AND,
                 OR,
                 XOR,
                 NOT,
                 SHIFT_UP,
                 SHIFT_DN
                }
```

XIT_FLOAT

XIT_FLOAT identifies the data as floating point. It is used in wire transport of some data (typically some client specified operands and some server returned data). It is never used in wire transport of images. An XIE float consists of three components:

```
XIT_FLOAT:
          [
            e_sign   - BOOLEAN,
            m_sign   - BOOLEAN,
            exponent - CARD8,
            mantissa - CARD16
          ]
```

E_sign is the exponent sign while m_sign is the mantissa sign. If a sign boolean is true, it indicates that the value is positive.

XIT_LINE_PROGRESSION

XIT_LINE_PROGRESSION defines the way in which scan lines are to be painted at presentation time. See the appendix on image data attributes & organization for more details.

```
XIT_LINE_PROGRESSION:  {90, 270}
```

XIT_MATHF

XIT_MATHF defines the valid mathematical functions that can be invoked through the mathematical process functions. They include:

```
XIT_MATHF
          {
           EXP,
           NATURAL LOG,
           LOG2,
           LOG10,
           SQUARE,
           SQUARE_ROOT,
          }
```

XIT_PIXEL_PROGRESSION

XIT_PIXEL_PROGRESSION defines the way in which pixels are to be painted along scan lines at presentation time. See the appendix on image data attributes & organization for more details.

```
XIT_PIXEL_PROGRESSION:  {0, 90, 180, 270}
```

XIT_POLARITY

XIT_POLARITY defines the way in which pixel data is to be interpreted at presentation time. In particular, it defines how data values are mapped into brightness. See the appendix on image data organization for more details.

```
XIT_POLARITY:  {ZERO-BRIGHT, ZERO-DARK}
```

XIT_PRED_TYPE

XIT_PRED_TYPE defines the dyadic predicate operators recognized by predicate protocol functions including:

```
XIT_PRED_TYPE:
               {
                LESS THAN,
                LESS THAN OR EQUAL TO,
                EQUAL TO,
                NOT EQUAL TO,
                GREATER THAN OR EQUAL TO,
                GREATER THAN
               }
```

XIT_STAT_TYPE

XIT_STAT_TYPE defines the types of statistical functions recognized in the protocol including:

```
XIT_STAT_TYPE:

              {
               MINIMUM,
               MAXIMUM,
               MEAN,
               STD_DEV,
               VARIANCE
              }
```

XIT_RESOURCE_TYPE

XIT_RESOURCE_TYPE is used to define the type of image extension server resource. It is used primarily during resource creation. The specific extension resource types are defined below. Details of their contents are found in the appendix on server resources.

```
XIT_RESOURCE_TYPE: {XIR_IOB, XIR_IDC, XIR_IPC}

    symbol              meaning

    XIR_IOB             image object
    XIR_IDC             domain  control context
    XIR_IPC             process control context
```

XIT_SPECTRAL_COMPONENT_MAPPING

XIT_SPECTRAL_COMPONENT_MAPPING defines the recognized 3 color space systems. As of this version, the only recognized spectral component mapping is RGB.

```
XIT_SPECTRAL_COMPONENT_MAPPING: {RGB}
```

XIT_FUNCTION_VECTOR

XIT_FUNCTION_VECTOR is used to convey information about server support for individual XIE protocol functions. Each bit of the XIT_FUNCTION_VECTOR represents a single protocol function in the image extension protocol. The index of the bit (zero relative) is the value of the protocol function number. A bit value of 1 indicates that the function is supported by the server, while 0 implies that it is not.

Chapter 5

PROTOCOL ERROR CODES

The image extension protocol makes use of both core protocol error codes and image specific extension protocol error codes.

5.1 Core Errors

The following errors are defined in the X11 core protocol specification. Refer to the X11 core protocol specification for their definition.

```
Access      Alloc       Atom        Colormap        Cursor
Drawable    Font        GContext    IDChoice        Implementation
Length      Match       Name        Pixmap          Property
Request     Value       Window
```

The image extension makes explicit use of the following core protocol errors:

- Access
- Alloc
- IDChoice
- Match
- Value

The image extension uses the core protocol Access error during the image data wire transport process. The error indicates that the client failed to establish the required transport context information and that wire transport of image data can not proceed until this is done.

The image extension uses the core protocol Alloc error for server resource allocation problems. The error indicates that the server was unable to allocate the resource. As with the core protocol, it is left to the implementors to decide as to the implications of this error for all the possible allocation problems which may arise within the extension server.

In extension protocol functions which create new server resources and assign to them client specified resource id's, the image extension uses the core protocol IDChoice (BAD_IDChoice) error to signal that the client desired new resource id is invalid or already in use.

In extension protocol functions which make use of existing server resources, the image extension uses the core protocol IDChoice (BAD_IDChoice) error to signal that the client specified resource id is invalid (wrong type of resource or nonexistent resource). This applies to image objects (IOB's), image domain control contexts (IDC's), and image processing control contexts (IPC's).

The image extension uses the core protocol Match error to signal mismatches between IOB constraint or chrominance state and the state required by the function at hand.

The image extension uses the core protocol Value error to signal attempts to assign values which are out of range to extension resource fields and when math errors occur.

All errors return a message with a common prefix as described in the Error Protocol Message Format section. Included in this information is the protocol function major opcode, minor opcode (extension function code), error code, type code, and the request sequence number. This information is followed by error code specific data described below.

5.2 Extension Errors

There are no image extension specific errors at this time.

Chapter 6

SERVER DATA MODEL

Data formats for wire transport of image data are fully specified in the protocol (next chapter). The natural tendency is to leave the particulars of image data representation on the server to server implementors. But, to some extent, the operation of certain extension protocol functions depend upon server image data representation. This chapter defines the bounds of a server image data model which will ensure some degree of operational consistency between various server implementations.

There are several dimensions to the server image data model. The first is the tradeoff between dynamic range/precision and processing speed/accuracy. The second is the relationship between server image data organization and spectral bands. The third is the ability to pipeline image data through multiple XIE server processing primitives.

6.1 Constrained vs Unconstrained

XIE has established the definition of an image at wire transport to be a two dimensional array of positive integral data which may be organized and compressed in various ways. The wire transport definition was chosen to force the application or client to reduce the dimensionality of its data to a least common denominator based upon the premise that the application understands its data model better than any one else.

The wire transport definition of image data is too restrictive for many computational purposes. A simple example is an algorithm which subtracts a constant from an image and then adds the constant back in. Under the positive integral constraint, this algorithm may not reproduce the original data. Another example of an algorithm that may suffer is one which takes the natural log of the data and then either an antilog or scaling of the result to fit within some dynamic range. The integral constraint throws away much information generated by the log function.

The server image data model must provide for a balance between representational flexibility (regarding the dynamic range and precision of the data) and computational efficiency. Typically, this is done by mapping the data directly onto one of the machine architecture's primitive data formats (integer or float). Differences between various server machine architectures regarding word size for integers and robustness of floating point support require XIE to adopt a logical rather than physical approach.

XIE adopts two logical image data representation models on the server: constrained and unconstrained. Constrained data will be positive integral data with limited dynamic range and fixed precision [zero based with a fixed precision (number of levels or number of bits per spectral component per pixel)]. Unconstrained data may be positive or negative and of dynamic range less than or equal to the maximum that the server implementor (typically machine architecture) can provide. As such, its dynamic range will be higher than that of constrained data. While float may be a natural choice for unconstrained data, server implementors are free to choose a long integer to represent unconstrained data. The process of converting from unconstrained to constrained data is one of making all the data positive, and with fixed number of quantization levels.

The client has substantial control over when and how constraints are applied. The client has the ability to modify the parameters which control how constraining is done (default constraint model). The client has the ability to query an IOB and determine its state relative to constraints. The client has the ability to explicitly force conversion to constrained state directly (through a "constrain" protocol function), or indirectly (after the application of process protocol functions).

The constrained/unconstrained model applies to all server functions which deal with image data (thus, constrained/unconstrained is meaningless for functions such as select events and query IDC which do not deal with image data).

Protocol functions which should not accept unconstrained data (wire transport, bitwise logical functions,...) will reject it (user is forced to explicitly constrain it). Protocol functions which are asked to output constrained data will apply the constraint function after their basic function is performed (ie. ensure maximum dynamic range & precision in the calculation and apply constraints on the way out).

For example, let the client specify a constraint model of 256 levels for a gray scale image, and let there be a test image which was sent across the wire with equal distribution of pixels in all 256 states (it is constrained). If the client asks for a constrained subtraction of 128 from the image and the current default constraint model is a hard clip within dynamic range, the resulting image is constrained and has half its pixels clipped to value 0, and the other half equally distributed between 0 and 128. If the client asks for an unconstrained subtraction of 128 from the image, the resulting image is unconstrained and has its pixels equally distributed between -128 and +128. After generation of this unconstrained image, if the client asks to mask out the low four bits, the server will first convert the unconstrained image to a constrained image, then perform the mask, and produce a constrained image.

Particulars of constraint functions are found in the following protocol sections:

- session services
  - establish operational defaults
  - query operational defaults
- radiometric services
  - apply constraints

The client should establish the default constraint model and parameters immediately after session connection.

6.2 Mono vs Poly Chromatic

While it may be possible to have client side libraries manage three IOB's as the constituent components of a color image, it is felt that this is too minimalistic a view of XIE color image support and that some color IOB data organizations and extended server function capabilities should exist for color image objects.

Earlier drafts recognized multiple color three spaces, and undoubtedly, in the future these three spaces (such as LUV and certain CIE spaces) will become significant to the image display situation on workstations. But, for now, XIE will focus on support only for R-G-B images. Thus, three space color conversion and color correction problems will remain on the client side of the wire (application level or toolkit level issues).

The impact upon the XIE protocol is the following:

- IOB's are typed regarding color space (mono or rgb only), and this information is queryable.
- For rgb images, all planes are constrained or unconstrained.
- For rgb images, the number of pixels-per-scanline and number-of-scanlines must be identical for each of the three bands.
- For rgb images, the number of bits-per-component may be different for each band.
- Various server function specify their operational capabilities for mono and rgb images.
- A function exists for the separation of rgb into three mono components.
- A function exists for the combination of three mono components into rgb.

Regarding spectral bands, it is suggested that server implementors keep spectral band data separate within an IOB. As such, an IOB for a monochromatic image (bitonal or continuous tone grayscale) would have one "band" type object (1 or n bits deep), while an IOB for a polychromatic (RGB) image would have three "band" type objects (r, g, and b bits deep respectively). Note that this is only a suggestion, and not a requirement. Clients should not assume or infer that this is the underlying image data organization on the server.

In the future, we will consider support for alternate color three spaces. This support will involve IOB color space state information which is queryable, and server implementation of three space color transforms as well as possible extensions to existing protocol functions.

6.3 Data Flow Pipelining

The X image extension protocol strives to be primitive yet flexible enough to support many higher level image rendition, enhancement, and display related operations. The understanding of these higher level operations and intelligence in wielding the low level extension protocol functions to achieve them has been left to an extension tool kit and the application itself. The X image extension does offer a mechanism to aid in organizing and executing a series of primitive protocol functions known as an image data processing pipeline.

Under the X image extensions, a pipeline is defined to be a single threaded, list of X image extension protocol functions through which image data flows sequentially. As such, the pipeline mechanism is intended to support sequential segmented flow of image data through multiple image extension process steps. These pipelines are driven by data availability (pipeline elements are automatically stalled when awaiting data - individual pipeline elements establish minimal required data availability for processing to occur). At any point in time, the pipeline may be partially or completely full of data with the implication that several pipeline segments may be "active" (able to process or processing data {in the case of multiple processors}) at the same time.

An example of such a pipeline might be one where as image data arrives from the client through transport calls, it moves through various rendition steps (perhaps scaling and tonal remapping) to display. Thus, as image data arrives at the server, it is streamed to the display, versus waiting for arrival of the complete image - then moving it completely through the rendition steps one at a time - then moving it through the display step.

There are several significant constraints on the pipelining mechanism. The first is that only a subset of the extension protocol functions may be used as pipeline elements (see summary of protocol appendix). No core protocol functions may be used in a pipeline. No protocol function which exchanges information across the wire may be used in a pipeline other than image data wire transport functions. Second, there are no pipeline process control functions (no test mechanisms, iteration mechanisms, or human interaction mechanisms for use within a pipeline). In essence, pipelines are sealed, straight through processing mechanisms. The mechanism for describing (assembling) pipelines is simplistic (vs reinventing complex language interpreters). As such, application and toolkit developers will need to "layout" or design pipelines on paper before hand working out all parameter binding issues and then hand build the calls to construct the pipelines. Note also that no pipeline "registry" mechanism exists and so pipeline definitions exist only within the bounds of an extension session.

Regarding the execution of pipeline processing elements, server implementors are free to use special accelerators and parallel processing to execute pipeline elements (just as they are to accelerate an extension protocol function outside of a pipeline). The pipeline mechanism does not provide any control over or understanding of these situations.

X image extensions will provide for definition, execution, and monitoring of segmented image data processing pipelines on the server (see resource and process control services).

Chapter 7

REQUESTS AND REPLIES

This section constitutes the body of the image extension protocol. It consists of protocol functions which are described in a uniform fashion (refer to syntax subsection). All X image extension protocol function names begin with the prefix XIEP_.

Note that like the core protocol, all image extension protocol functions are asynchronous in nature. Clients requiring synchronous behavior should use the core protocol's approach to achieving synchronous behavior.

Extension protocol requests follow the byte ordering established at core protocol connection time with regard to all 16 and 32 bit quantities. They also comply with maximum-request-length as specified in the core protocol connection.

7.1 Session Services

Image extension sessions occur within the bounds of core protocol sessions. The image extension protocol provides per client extension server session initialization and termination functions. These allow the client to initialize global extension session server information and allow the extension server to clean up on session termination. The session services include:

- XIEP_INIT_SESSION - begins a client-extension server session
- XIEP_SET_DEFAULTS - establish/modify operational defaults
- XIEP_QUERY_DEFAULTS - query operational defaults
- XIEP_TERM_SESSION - ends a client-extension server session

7.1.1 Initialize Session

```
XIEP_INIT_SESSION

        protocol_major_version:         CARD16
        protocol_minor_version:         CARD16

     =>

        success:                        BOOL
        protocol_major_version:         CARD16
        protocol_minor_version:         CARD16
        service_class:   CARD32
        supported_functions:            XIT_FUNCTION_VECTOR
        length:                         CARD16

     service class value:       0
     errors:                    none
     events:                    none
```

XIEP_INIT_SESSION must be invoked by a client as its first image extension protocol request as it causes the server to perform initialization on a per client basis. In addition, as with the core protocol connect request, the function returns information which is useful to the client.

Success is simply a statement of relationship between the server's protocol version, and the client request. TRUE implies that the server's protocol version is greater than or equal to the client's request and that service will be granted. FALSE implies the opposite.

Protocol version numbers in the image extension connection request refer to the image extension protocol. They follow the core protocol rules: major version changes for incompatibility reasons; minor version changes for small, upward compatible modifications.

Service_class is a mechanism for communicating the level of service provided by the particular server. See the concepts document for an overview of service class partitioning of XIE server capabilities.

Supported_functions is a mechanism for communicating exactly which protocol functions the server recognizes/supports. A value of 1 for a bit in the XIE_FUNCTION_VECTOR implies that the protocol function is recognized and supported by the server. Specialized hardware support is not implied as this is communicated elsewhere. While the mechanism allows the server to describe support at the individual protocol function level of granularity, it is strongly recommended that servers implement all functions in a service class (see concepts document).

Length is the amount of additional data which follows in units of 4 bytes. Additional data received if connection accepted includes...

```
        server_hardware:          HARDWARE
where:

     HARDWARE: {
                  accelerators:                  XIT_ACCELERATOR_VECTOR
                  screens:                       LIST_of_SCREEN
                }

     SCREEN:    {
                  screen_number:                 CARD8
                  accelerators:                  XIT_ACCELERATOR_VECTOR
                }
```

7.1.1.1 Per Server information

Server information includes a description of the particular server hardware image capabilities, and a list of descriptions of the various display screens image capabilities. No core server session redundant information is included. Therefore, a complete picture of the hardware and screen(s) derives from the union of core server and extension server connection request responses.

Accelerators provides the client information about which image extension protocol functions have hardware acceleration available across all screens. This information is provided to guide imaging applications with regards to what they might reasonably expect from a server. Accelerator information is formatted as a support vector. In particular, there is a bit for every protocol function whose value indicates the state of hardware support for that function on the server. A '0' implies no direct hardware acceleration (ie. the protocol function is implemented through software on the server host processor), while a '1' implies specific accelerator support (ie. a processor other than the server host processor is used to accelerate the function).

7.1.1.2 Per Screen Information

The screens section of the reply addresses screen specific imaging hardware , screen specific accelerators, and display controls.

Screen number is the number by which client rendition requests identify the screen they are to affect.

Accelerators provides the client information about which image extension protocol functions have hardware acceleration available on this screen. The format is identical to the hardware accelerators field.

7.1.1.3 Relationship to Core Session Information

At the time of initial connection to the core X server, the client receives information back from the core X server. This includes connection and "per server" information. The availability of this information to the client precedes the availability of X Image Extension initialization information as the core X connection precedes the X Image Extension initialization.

The connection information used by a client of the X Image Extension consists of a combination of information which comes from the core connection and the image extension initialization. Some core connection information is applicable to the image extension session. This information is not replicated at image extension initialization time. Other core connection information is not applicable to the image extension model, and is superseded by information provided at image extension initialization time. Finally, there is image extension specific information which was not found in the core connection information.

The table below summarizes the core connection information and its usefulness in an image extension session.

core information name	meaningful	type
. success	no	
. protocol-major-version	no	
. protocol-minor-version	no	
. vendor	yes	STRING8
. release-number yes CARD32		
. release-number	no	
. resource-id-base	yes	CARD32
. resource-id-mask	yes	CARD32
. bitmap-format-scanline-unit	yes	{8,16,32}
. bitmap-format-scanline-pad	yes	{8,16,32}
. bitmap-format-bit-order	yes	{LSBfirst,MSBfirst}
. pixmap-formats	yes	{ListofFormat}
. roots	yes	{ListofScreen}
. keyboard	no	
. pointer	no	
. motion-buffer-size	no	
. maximum-request-length	yes	

Within roots, useful (applicable) information includes: root, width and height, default_color_map, white_pixel, black_pixel, min_installed_maps, max_installed_maps. All other information is not applicable.

7.1.2 Set Operational Defaults

```
XIEP_SET_DEFAULTS

        constraint     - see below -

     =>

        none

     service class value:       0
     errors:                    VALUE
     events:                    none

where:

     constraint includes:

        [
         constraint_model       CARD8
         bits_pel_component     LISTofCARD8
        ]
```

XIEP_SET_DEFAULTS is used to modify operational defaults used by the extension server. Operational defaults involve information which controls the operation of the extension server, is knowable/modifiable by the client, and is not embodied in a server resource. It may be invoked at any time to change the operational defaults.

Constraint defaults involve information used by the server in converting image data between unconstrained and constrained states. The information includes the constraint_model which defines how image data is constrained and bits_pel_component (bits per pixel per spectral component) which determines the number of quantization levels of the resulting constrained data.

The terms floor and ceiling will be used to described each of the constraint models. Floor refers to the smallest positive integral value (base) allowed after constraining the data - for XIE, floor is fixed at 0 (zero). Ceiling is the largest positive integral value allowed after constraining the data - for XIE it is set to ((2**n) - 1) where n is the number of bits per pixel per component (ie. ceiling is an integral power of two, and may be different for each spectral component). Constraint models include:

1. HARD CLIP
   The data is first clipped such that all values below the floor (negative) are set to the floor value, and all values greater than the ceiling are set to the ceiling value. Then, all values are mapped to the nearest integer within the floor-ceiling range by rounding.
2. CLIP SCALE
   The data is first clipped such that all values below the floor (negative) are set to the floor value, and all values greater than the ceiling are set to the ceiling value. Then, all values are scaled to fit within the ceiling-floor range (make maximum use of the available dynamic range. Finally, all values are mapped to the nearest integer within the floor-ceiling range by rounding.
3. HARD SCALE
   The data is first shifted upwards so that all values are greater than or equal to the floor value (positive). Then, all values are scaled to fit within the ceiling-floor range. Finally, all values are mapped to the nearest integer within the floor-ceiling range by rounding.

Note that XIE supports only linear quantization models (scaling).

If any of the constraint values are out of range, or incompatible, a VALUE error occurs.

7.1.3 Query Operational Defaults

```
XIEP_QUERY_DEFAULTS

        none

     =>

        constraints     - see preceding section -

     service class value:       0
     errors:                    none
     events:                    none
```

XIEP_QUERY_OP_DEFAULTS is used to query the current state of the extension server's operation defaults. Operational defaults involve information which controls the operation of the extension server, is knowable/modifiable by the client, and is not embodied in a server resource.

See preceding section on establishing operational defaults for details of the information returned.

7.1.4 Terminate Session

```
XIEP_TERM_SESSION

        none

     =>

        none

     service class value:       0
     errors:                    none
     events:                    none
```

XIEP_TERM_SESSION is used by a client to terminate an image extension session. It must be invoked by a client as its last image extension protocol request as it allows the server to perform clean up on a per client basis. The extension server deletes all client specific resources. A client may choose to continue with a core session after terminating an extension session.

7.2 Resource Services

Image extension resources are extension server objects known to the client which contain information used by the extension server in carrying out various protocol functions. Extension resource class services provide for the management of these extension server objects by the client. Extension resource protocol functions address resource creation, deletion, modification, query, and copy. In addition, extension resource services provide for moving image data between core and extension resources. Resource protocol functions include:

- XIEP_CREATE_RESOURCE - create a new server resource object
- XIEP_COPY_RESOURCE - copy contents from source to destination resource
- XIEP_DELETE_RESOURCE - delete an existing server resource object
- XIEP_QUERY_RESOURCE - query an existing server resource object
- XIEP_IMPORT_IMAGE - create a new server IOB resource object
- XIEP_EXPORT_IMAGE - create a new server PIXMAP resource object

7.2.1 Resource Creation

```
XIEP_CREATE_RESOURCE

        resource_id             {XIT_IOB,XIT_IDC,XIT_IPC}
        resource_type           XIT_RESOURCE_TYPE
        resource_data           - see below -

    =>

        none

    service class value:         - see below -
    errors:                      ALLOC, IDChoice, VALUE
    events:                      none


where resource_data is one of the following:

        {
         image_object_resource_data,
         process_control_context_resource_data,
         domain_control_context_resource_data
        }
```

Server resource creation causes the server to allocate a new resource and to fill it with information passed through the function's parameters.

Resource_id is the client specified id to be assigned to the new resource. Resource_type specifies what type of extension server resource is to be created., (image object resource, process and domain control object resources). Resource_data is type specific and is described in the following subsections.

All extension server resources have their id's selected by the client (same model as for all core resources). The client must use the core session connection assigned resource-id-base and resource-id-mask in generating id's for image extension resources. As specified in the core protocol "The resource_id_mask contains a single set of contiguous bits (at least 18); the client

allocates resource ids by choosing a value with (only) some subset of these bits set, and ORing it with the resource_id_base. Only values constructed in this way can be used to name newly created resources over this connection. Resource ids never have the top 3 bits set. The client is not restricted to linear or contiguous allocation of resource ids. Once an id has been freed, it can be reused, but this should not be necessary. An id must be unique with respect to the ids of all other resources, not just resources of the same type."

The service class of the resource creation function depends upon what type of resource is being created. This is discussed in each of the following subsections.

Inability of the server to allocate the new resource results in an ALLOC error. Choice of a resource id which is already in use or is out of range generates an IDChoice error. Invalid value specification of any image attributes results in a VALUE error. If any error occurs, the object is not created.

7.2.1.1 Image Object Resource

```
Image_Object_Resource_Data

      [
        pixels-per-scanline            CARD32
        number-of-scanlines            CARD32
        aspect_ratio                   XIT_FLOAT
        number-of-spectral-components  CARD8
        bits_per_component             LISTofCARD8
        spectral-component-mapping     XIT_SPECTRAL_COMPONENT_MAPPING
        pixel_progression              XIT_PIXEL_PROGRESSION
        line_progression               XIT_LINE_PROGRESSION
        polarity                       XIT_POLARITY
      ]
```

IOB's represent revisable form images (see server resources and image data attributes and organization appendices). They consist of image attributes and image data.

Image objects can come into existence in several ways: as the result of a client request to create a new one; as the result of an import protocol request; and as the result of a processing protocol request. XIEP_CREATE_RESOURCE with type image object is used by a client to create (allocate) a new server image object resource and as such, represents the first of the three ways of creating new image objects.

Clients typically create new image objects just prior to populating them with image data via transport protocol functions. Both primitive and presentation attributes are communicated from client to server during image object creation. The primitive attributes are strongly bound to the data as they describe the most fundamental organizational information. The presentation attributes are less strongly bound as they provide some insight into the original presentation of the data.

```
primitive_attributes include:

      pixels-per-scanline
      number-of-scanlines
      number-of-spectral-components
      bits_per_component
      spectral-component-mapping

presentation_attributes include:
```

```
pixel_progression
line_progression
aspect_ratio
polarity
```

Note that while most client accessible server resource object fields can be initialized at creation time, there may be fields in image objects which are initialized and maintained by the server and are only queryable by the client (eg. constraint information)

Pixels-per-scanline, number-of-scanlines, number-of-spectral-components, and spectral-component-mapping are all primitive image attributes described in the appendix on image data attributes and organization. While bits_per_component is an artifact of image data wire transport (more so than a primitive attribute), it is communicated with primitive attributes at IOB creation time to facilitate IOB resource allocation (memory).

Pixel_progression, line_progression, aspect_ratio, and polarity are all original image presentation attributes described in the appendix on image data attributes and organization.

IOB's created by the create resource function begin their life with state constrained. This is consistent with their only initial use (receiving image data from across the wire). IOB image data is initialized at creation time to the value zero (0). This is done to ensure that all image locations begin with meaningful data. This is particularly important in that image data transport overlays the default values and while stream image data transport must by definition overlay all of the data locations, random access tiled image data transport may or may not.

IOB's are potentially very expensive server resources due to their size. Care should be exercised by the client regarding which and how many resources are allocated.

7.2.1.2 Domain Control Context Resource

```
domain_context_resource_data

        idc_type        XIT_IDC_TYPE
        idc_data       - see below -


Where idc_data depends on the idc_type:

       . for ROI's, idc_data is the following sequence:

            [
              ul_x              CARD32
              ul_y              CARD32
              width             CARD32
              height            CARD32
            ]

       . for CPP's, idc_data is the following:

            [
              ul_x              CARD32
              ul_y              CARD32
              cpp_iob           XIT_IOB
            ]

       . for TMP's, idc_data is the following:
```

```
    [
      center_x            CARD32
      center_y            CARD32
      data_count          CARD32
      tmp_data            LISTofTMP_ENTRY
    ]

where:

    TMP_ENTRY is:

  [
    x_index             CARD32
    y_index             CARD32
    value               XIT_FLOAT
  ]
```

Domain context resources are used to focus various classes of server based image processing onto a subset of the pixels that are part of an IOB, and to act as operands in certain image area operations.

IDC_type determines if the IDC is to be an ROI, CPP, or TMP. IDC_data specifies the bounds of an ROI, identifies a bit plane (an IOB which is one bit per pixel) to be used as a CPP, or the locations and values of the template.

There are three forms of IDC: the region of interest or ROI, the control processing plane or CPP, and the template or TMP. An ROI defines a closed, rectangular subregion. When using an ROI type IDC, all pixels within (border included) the ROI are subject to processing. Individual operators define how boundary conditions are handled. A CPP defines a potentially fully disconnected set of locations such that pixels at these locations are subject to processing. CPP's do this by representing all pixel locations with a single bit, and using the state of the bit to control whether processing occurs or not. A TMP defines the shape and values of a template or mask used in area operations. TMP's maybe fully or sparsely populated two dimensional entities. Individual protocol processing functions define which forms of IDC are acceptable.

All three forms of IDC (ROI, CPP, TMP) are spatially registered with the image object data (IOB) through the image object's implicit two dimensional coordinate system. ROI and CPP's are spatially registered by the values of their upper left coordinates which specify the location in image object coordinates of the upper left corner. In the case of ROI's and CPP's, their x and y extents must be less than or equal to that of the IOB they will be used with at operation time.

While the X image extension protocol does not provide for object recognition, it does allow clients to process and convert an IOB into a CPP via the boolean operators in enhancement services section. With these services, a client can generate a CPP for pixels with the same intensity or color, or more complex identity relationships. Applications such as chroma keying may be possible with these services.

The spatial registration of a template (TMP) with the area image data surrounding a pixel is established by the centerx and centery data. All templates are specified in the positive x and positive y quadrant of a cartesian coordinate system. The centerx, centery data positions the location of the data pixel (central pixel) relative to the template. For example, the following hollow cross filter would be represented in the protocol as shown below:

```
                                -7.00

                        +3.00         -0.75

                                +1.66

        entries     4
        centerx     1
        centery     1

        x_index    1      2       1      0
        y_index    0      1       2      1
        value     +1.66  -0.75   -7.0   +3.0
```

Note that asymmetric templates are possible, and there is no need for all or even the center point template value to be defined (sparse definitions). Note also that the size of the list length entry (first field in the list) for tmp_data is CARD16 (rather than the default CARD8) to allow for large templates.

Creation of IDC resources is considered to be a service class 1 function as it is central to server image processing operations.

7.2.1.3 Process Control Context Creation

```
Process_Control_Resource_Data

      [
        name                          LISTofSTRING8
        inputs                        PIPE_IOT
        outputs                       PIPE_IOT
      ]

where:

  PIPE_IOT includes:

      [
       number                         CARD8
       formal_parm_types              LISTofXIT_PARMS
      ]
```

Process context resources represent server pipeline resources. They act as both a pipeline template and when executed, a pipeline instantiation.

Name is simply a descriptive string assigned by the client. It is used by pipeline segments to refer to global inputs and outputs, as well as by the server when a pipeline is nested. The IPC (pipeline) inputs and outputs represent formal parameter declarations for a process. In essence, they allow the client to type specify all inputs into the front end of a pipeline, and all outputs produced and returned from the tail end of a pipeline. Both input and output parameters are implicitly numbered by the server beginning with zero (0). This numbering is used to refer to the global inputs and outputs.

Note that the size of the list length entry (first field in the list) for formal_parm_types is CARD16 (rather than the default CARD8) to allow for large templates.

IPC's may be built, copied, queried, deleted, executed, and terminated. Creation of a resource of type IPC is the first step in defining an executable pipeline. After this step, the client must build the pipeline by sequentially adding elements to the pipeline (XIEP_ADD_SEGMENT). When the pipeline construction is complete, execution may begin (XIEP_EXECUTE_PIPE).

As an IPC represents both a pipeline template and instantiation, only one copy of any IPC may be executing at a time. Note that pipelines may be nested within other pipelines.

7.2.2 Resource Copy

```
XIEP_COPY_RESOURCE

    src_resource_id       {XIT_IOB,XIT_IDC,XIT_IPC}
    dst_resource_id       {XIT_IOB,XIT_IDC,XIT_IPC}
    resource_type         XIT_RESOURCE_TYPE

    =>

    none

    service class value:  0
    errors:               IDChoice, ALLOC, MATCH
    events:               none
```

XIEP_COPY_RESOURCE is used to copy the contents of a source resource into a destination resource. The client specifies the source resource, and the resource id to be used for the new (destination) resource, as well as the resource type. The server checks to see that the source resource is of the type specified, allocates a new (destination) resource of the same type as the source, and fills it with source resource contents.

Src_resource_id is the id of the source resource (copy will occur from this resource). Dst_ resource_id is the client assigned id of the new (destination) resource (copy will occur to this resource). Resource_type is the type of the source resource. Upon completion, the destination resource is an exact copy of the source resource.

It is left to the server implementor to determine if XIEP_COPY_RESOURCE will operate on the image data in an IOB (ie. if it will transfer both image attributes and data from source to destination IOB such that no two IOB's ever share the same image data, or if IOB's can share image data).

Attempts to access source resources which don't exist or to use a resource id for the destination resource which is already in use generates an IDChoice error. Inability to allocate the destination resource results in a ALLOC error. A mismatch between the source resource type and the specified resource type generates a MATCH error.

7.2.3 Resource Delete

```
XIEP_DELETE_RESOURCE

        resource_id     {XIT_IOB, XIT_IDC, XIT_IPC}
        resource_type   XIT_RESOURCE_TYPE

        =>

        none

    service class value: 0
    errors:              IDChoice, MATCH
    events:              none
```

XIEP_DELETE_RESOURCE is used by a client to delete (deallocate) server resources.

Resource_id identifies the specific resource the client wants deleted. Resource_type is the type of the resource to be deleted. Deletion of IOB's deletes the image data. As under the core protocol, the management of shared resources is an application problem.

Attempts to delete a resource which is nonexistent results in an IDChoice error. A mismatch between the source resource type and the specified resource type generates a MATCH error.

7.2.4 Resource Query

```
XIEP_QUERY_RESOURCE

        resource_id             {XIT_IOB, XIT_IDC, XIT_IPC}
        resource_type   XIT_RESOURCE_TYPE

    =>

        resource_data           - see below -

    service class value:         - see resource creation -
    errors:                      IDChoice, MATCH
    events:                      none

where resource_data is one of the following:

        {
         image_object_resource_data,
         process_control_context_resource_data,
         domain_control_context_resource_data
        }
```

XIEP_QUERY_RESOURCE returns the contents of all client readable fields of the server resource. All image extension server resources other than the image data itself (within the IOB) can be queried. Image data query is accomplished with a different protocol function.

Resource_id is the id of the resource the client wishes to query. Resource_type is the type of the resource to be queried.The data returned depends upon the type of resource being queried and is described in the following subsections.

The service class of the resource query function depends upon what type of resource is being queried. This is discussed in each of the following subsections. See the appendices on on server resources for more details of the attributes.

Querying a non existent resource generates an IDChoice error. A mismatch between the source resource type and the specified resource type generates a MATCH error.

7.2.4.1 Image Object Query

```
Image_Object_Resource_Data

        constraint_model              CARD8
        pixels-per-scanline           CARD32
        number-of-scanlines           CARD32
        aspect_ratio                  XIT_FLOAT
        polarity                      XIT_POLARITY
        pixel_progression             XIT_PIXEL_PROGRESSION
        line_progression              XIT_LINE_PROGRESSION
        spectral-component-mapping    XIT_SPECTRAL_COMPONENT_MAPPING
        number-of-spectral-components CARD8
        bits_per_component            LISTofCARD8
```

The function returns the contents of all of the queryable IOB attribute fields. It does not return the image data itself (see transport protocol functions).

Note that constraint information is never directly settable by the client, but is queryable. The constraint_model and constrain_bits_per_component fields reflect the state of the default operational constraint model when the IOB was last constrained.

The image object query can be viewed as returning three classes of information:

```
        primitive_attributes           - see below -
        presentation_attributes        - see below -
        constraint_info                - see below -

where:

  primitive_attributes include:

        pixels-per-scanline
        number-of-scanlines
        number-of-spectral-components
        spectral-component-mapping

  presentation_attributes include:

        pixel_progression
        line_progression
        aspect_ratio
        polarity

  constraint_info includes:

        constrained
        constraint_model
        bits_per_component
```

7.2.4.2 Domain Control Context Query

```
domain_control_context_resource_data

    idc_type        XIT_IDC_TYPE
    idc_data       - see below -

Where idc_data depends on the idc_type:

       . for ROI's, idc_data is the following sequence:

              ul_x              CARD32
              ul_y              CARD32
              width             CARD32
              height            CARD32

       . for CPP's, idc_data is the following:

              ul_x              CARD32
              ul_y              CARD32
              cpp_iob           XIT_IOB

       . for TMP's, idc_data is the following:

            center_x            CARD32
            center_y            CARD32
            data_count          CARD32
            data                LISTofTMP_ENTRY

         where:

            TMP_ENTRY is a triplet:
```

```
x_index             CARD32
y_index             CARD32
tmp_data            XIT_FLOAT
```

7.2.4.3 Process Control Context Query

```
process_control_context_resource_data

      [
        name                            LISTofSTRING8
        inputs                          PIPE_IOT
        outputs                         PIPE_IOT
        running                         BOOL
        segments                        LISTofSEGMENT
      ]

where:

  PIPE_IOT includes:

      [
       number                           CARD8
       formal_parm_types                LISTofXIT_PARMS
      ]

  SEGMENT includes:

      [
       name             LISTofSTRING8
       function         XIT_XIEFUNCTION
       parm_bindings    LISTofBINDING
       state            {processing, stalled}
      ]

where

  BINDING is:

    [
     name                           STRING8
     number                         CARD32
    ]
```

The function returns the contents of all of the queryable IPC fields.

Running is a boolean which indicates if the pipeline has been instantiated and is under execution, or is in the template state. Refer to the sections on XIEP_CREATE_RESOURCE (type IPC) and XIEP_ADD_SEGMENT for more details on other fields. If running is true, then the state field in the segments list elements is meaningful and describes the current state of each pipeline element (either processing, or stalled awaiting data).

7.2.5 Import/Export Services

Import/Export class services provide for the direct movement of image data and attributes between extension package objects (IOB's) and core server objects (drawable). The term "direct" means that other than issuing the protocol request, no further client intervention is required (the image data does not have to move back and forth across the wire). These services are provided in order to allow some server level mechanism for data exchange between the image extension and the core as well as other extension packages.

7.2.5.1 Import Image

```
XIEP_IMPORT_IMAGE

        iob_id          XIT_IOB
        drawable        DRAWABLE

        =>

        none

    service class value: 0
    errors:   DRAWABLE, MATCH, ALLOC, IDChoice
    events:                     none
```

XIEP_IMPORT_IMAGE transfers image data and attributes from an existing server drawable into a new server IOB.

IOB_id is the id to be assigned to the new IOB. Drawable is the id of a core server drawable resource that the image data and attributes will come from. The drawable may be either a pixmap or a window.

The server creates a new IOB to receive the image data and attributes. The client accessible IOB attributes (primitive and presentation) derive from information associated with the drawable. Data formatting information (organization and encoding) which is involved in wire transport plays no visible role in the process.

Note that all drawable image data is imported (no subregion specification is allowed and no import processing such as masking is allowed). Clients wishing to import only a portion of a drawable must extract the portion into a new drawable before using XIEP_IMPORT_IMAGE or crop the IOB after import. Clients wishing to process (such as mask) the data should use X Image Extension functions to do so once the data has been imported into an IOB.

For windows, pixels-per-scanline and number-of-scanlines derive from window width and height respectively, while number-of-spectral-components and spectral-component-mapping are inferred from the visual type of the window.

For pixmaps, pixels-per-scanline and number-of-scanlines derive from pixmap width and height respectively, while number-of-spectral-components and spectral-component-mapping are inferred from (TBS).

In all cases, the resulting IOB is in the constrained state.

In either case (pixmap or window), the default presentation attributes associated with the image data in the IOB (pixel-progression, line-progression, aspect-ratio, polarity, and grid-type) derive from server private knowledge of the display device.

If drawable does not specify a valid drawable, then a Drawable error occurs. If the drawable is an input only window, a Match error occurs. If the client specified IOB id is already in use, an IDChoice error occurs. If the server can not allocate the IOB, an ALLOC error occurs.

7.2.5.2 Export Image

```
XIEP_EXPORT_IMAGE

        iob_id          XIT_IOB
        drawable        DRAWABLE
        gc              GCONTEXT
        src_x           INT32
        src_y           INT32
        width           CARD32
        height          CARD32
        dst_x           INT32
        dst_y           INT32

        =>

        none

    service class value:        0
    errors:   IDChoice, MATCH
    events:                     none
```

XIEP_EXPORT_IMAGE combines image data from a rectangular region of an existing server IOB with data in a rectangular region of an existing server DRAWABLE.

IOB_id is the id of the source image object. Drawable is the id of the receiving DRAWABLE (WINDOW or PIXMAP). GC_id is the graphic context used to provide information used when filling the drawable. Src_x and src_y are the location of the upper left hand corner of the source IOB data. Width and height define how much data to transfer. Dst_x and dst_y are the location of the upper left hand corner of the destination location in the receiving DRAWABLE.

XIEP_EXPORT_IMAGE is similar in many aspects to the core X copy_area function. The similarities include transfer of image data into a server resource with control over subarea and combination of source and destination data, as well as the requirement that the data format must match that of the receiving resource (no format conversion or mismatches supported or allowed).

XIEP_EXPORT_IMAGE will only transfer data between and IOB and a DRAWABLE. It is not capable of reconciling differences between the IOB data and the pixmap formats. In other words, the process of reconciling the differences between what have been referred to as revisable form images and what the display device can show (this is reflected in DRAWABLE formats) must occur prior to export. This reconciliation process is what is referred to as rendition and must be performed before export.

Thus, prior to use of export_image, the IOB must be constrained and the client should apply various rendition steps/operations (such as scale, crop, ...) to ensure that the IOB image data is compatible with the receiving DRAWABLE.

If iob_id or drawable specify invalid (nonexistent) resources, an IDChoice error occurs. If an attempt is made to export an IOB into a pixmap such that the IOB is unconstrained or its data is incompatible with the data formats which a DRAWABLE can contain (see core X11 protocol connection function), then a MATCH error occurs.

7.3 Transport Services

Transport class services provide the means for exchange of image data between client and server. Transport protocol functions include:

- XIEP_SET_TRANSPORT - establishes the required transport context information
- XIEP_ABORT_TRANSPORT - aborts transport & clears transport context information
- XIEP_GET_TILE - transports random tiles of uncompressed image data from server to client
- XIEP_PUT_TILE - transports random tiles of uncompressed image data from client to server
- XIEP_GET_STREAM - transports a stream of compressed or uncompressed image data from server to client
- XIEP_PUT_STREAM - transports a stream of compressed or uncompressed image data from client to server

7.3.1 Overview of Data Transport

This section addresses the overall image data transport process, transport modes, transport data formats, transport synchronization, and transport granularity & atomicity. Details of these subjects are found within the data transport protocol definitions.

7.3.1.1 Transport Process

In client->server transfers, the client usually begins by creating a new image object (IOB) using XIEP_CREATE_RESOURCE and establishing primitive and presentation attributes as it does. Note that while client->server transport is usually done to a new (empty) IOB, an existing IOB (one which already contains image data deriving from a prior transport or processing operation) may be used to receive the transport image data.

The client then proceeds to establish the transport context (XIEP_SET_TRANSPORT) which controls the format of image data on the wire for this transport session to the specified IOB. Finally, the client transfers the image data from its buffer to the server IOB through one or more transport calls (XIEP_PUT_TILE or XIEP_PUT_STREAM). The transport session is completed when a final flag is set in one of the transport functions, or when the transport is aborted via XIEP_FREE_TRANSPORT.

In server->client transfers, the IOB already exists and is filled with image data. In this case, the client proceeds to establish the transport context (XIEP_SET_TRANSPORT) which controls the format of image data on the wire for this transport session from the specified IOB. Then the client transfers the image data from the server IOB to its buffer through one or more transport calls (XIEP_GET_TILE or XIEP_GET_STREAM).

Due to size limitations on protocol messages and/or client preferences, transport may involve one or many transport protocol function invocations.

7.3.1.2 Transport Modes

Image data transport is divided into two modes: random access tiled, and sequential access stream.

Random access tiled transport is designed to allow exchange of uncompressed tiles of image data between client buffer and IOB. The client specifies both tile size and IOB relative location on a transport function call basis. Tiles may be as small as a single pixel, or as large as the transport protocol message can handle.

Sequential access stream transport is designed to allow exchange of either uncompressed image data or compressed image code information in a sequential stream format. No random access is allowed in this mode.

The use of image data compression in sequential access stream transport involves a tradeoff between processing and communications bandwidth. The availability of processing power (including specialized compression/decompression engines), and the available bandwidth between client and server should both factor into the choice between compressed and uncompressed image data transport.

When compression is used with sequential access stream mode, image code information rather than image data is transported. The image can be recovered (with varying degrees of fidelity based upon the compression scheme) from the code information. The code information and transport both depend upon the compression scheme.

Client to server image data transport may occur to either a preexisting or a newly created IOB. As data arrives (uncompressed tiles, uncompressed stream, or compressed data which the server immediately decompresses), it overlays the initial data (default value = zero, or prior image). No combination of transport data (image data or code information) with preexisting IOB data other than simple overwrite is provided. This does not exclude complex assembly of IOB data through transport (such as aggregating and assembling bit planes in Bit_Interleaved_by_Plane format).

Both tile and stream (uncompressed and compressed) transport are applicable to all classes of digital images (bitonal, continuous tone monochromatic or gray scale, and continuous tone polychromatic or color).

7.3.1.3 Transport Data Formats

Whether random access tile or sequential access stream (uncompressed or compressed) transport will be used, the client must choose between three underlying PCM data organizations: Band_Interleaved_by_Pixel, Band_Interleaved_by_Plane, and Bit_Interleaved_by_Plane (see appendix for details). These data organizations define how the bits which represent the various spectral bands are organized.

In tile and stream uncompressed transport, this PCM format describes the actual data transport wire format. In stream compressed mode, compression is applied to the data after it is organized in one of the three ways. When transporting image data in the stream compressed mode, the client must choose the compression scheme. The valid combinations of compression schemes and underlying image data organizations are discussed below.

The transport functions which follow refer to a concept known as the "data plane" , or simply "plane". In fact, the atomic transport functions do not concern themselves with the meaning of the image data (at the level of it being a bitonal image, or a grayscle image, or a color image), but rather, they concern themselves with the assembly/disassembly of image data into transport buffers, and compression. The mapping between "types of images" (bitonal, continuous tone monochromatic, and continuous tone color), component space organization (band-interleaved-by-pixel, band-interleaved-by-plane, bit-interleaved-by-plane), and planes is as follows: ** tbs - table **

In any case, only positive integral values (image data or bit string codes) flow across the wire. There is no provision for exchange of negative integral image data, real valued (float) image data or complex valued image data on the wire even though such data (real valued) may exist on the server (see server data model). In terms of either client->server or server->client image data transport, the server side IOB must be of state constrained.

Flexibility in overall transport mode (tile or stream), underlying image type (bitonal monochromatic, continuous tone monochromatic or gray scale, and continuous tone trichromatic or color), underlying PCM image data organization (band_interleaved_by_pixel, band_ interleaved_by_plane, and bit_interleaved_by_plane), stream mode compression scheme (PCM, G4, DCT), and all or single plane at a time transport leads to an enormous number of possible transport configurations. Fortunately, some image data organizations are identical (degenerate) for some image types, and some compression schemes are inapplicable for some image data organizations. The following table is intended to give the reader an overview of the valid transport configurations. The all or single plane option is left out of the table and dealt with explicitly in the transport functions themselves. Degenerate image data organizations are grouped by surrounding brackets {}.

MODE	IMAGE_TYPE	IMAGE_ORGANIZATION	COMPRESSION
Tile			
	Bitonal	{band_interleaved_by_pixel band_interleaved_by_plane bit_interleaved_by_plane}	n/a
	Grayscale	{band_interleaved_by_pixel band_interleaved_by_plane}	n/a
		{bit_interleaved_by_plane}	n/a
	Color	{band_interleaved_by_pixel}	n/a
		{band_interleaved_by_plane}	n/a
		{bit_interleaved_by_plane}	n/a
Stream			
	Bitonal	{band_interleaved_by_pixel band_interleaved_by_plane bit_interleaved_by_plane}	PCM, G4
	Grayscale	{band_interleaved_by_pixel band_interleaved_by_plane}	PCM, DCT
		{bit_interleaved_by_plane}	PCM
	Color	{band_interleaved_by_pixel}	PCM, DCT
		{band_interleaved_by_plane}	PCM, DCT
		{bit_interleaved_by_plane}	PCM

Note that while DCT compression of bitonal image data is possible, and both G4 and DCT compression of bit_interleaved_by_plane data of grayscale and color images is possible, neither are proposed at this time. The need for and value of these schemes will be considered in the future.

7.3.1.4 Transport Granularity & Atomicity

Both the tile and stream (uncompressed and compressed) transport protocol functions are limited in terms of the amount of data they can move by the length field in the standard (core X11) protocol message. As images (in a client's buffer or in a server IOB) may be much larger than the length field limit, it may require multiple transport protocol calls to move the complete image across the wire.

The extension protocol transport functions allow the client to construct or assemble all of the data which will make up a single image (IOB) with multiple transport requests. In particular, there is a "final" flag in both tile and stream (uncompressed and compressed) transport protocol functions which is used to signal completion of transport and assembly of the data which makes up an image. In the case of tile image data transport, this flag is used when the client has populated through transport all tiles it wishes to in the IOB. In the case of stream (uncompressed or compressed) image data or code information transport, this flag is used when the client has sent the last sequential stream of image data or code information. In the tile case, a single flag applies to all planes, while in the stream case, a separate final flag applies to each plane.

The transport parameters associated with an IOB (XIEP_SET_TRANSPORT) are locked during transport of intermediate (non final) segments. In other words, once transport to or from an IOB begins, the client can not modify the transport parameters until a final transport (final flag = true) occurs.

The client has control over the size of each transport request, and may change the size on a request by request basis. This is intended to support applications which may be acquiring/disposing of image data (from disks or networks) in small pieces (including due to highly limited virtual address space / physical memory space), as well as applications making use of X Image Extension Pipelines where transport is part of the pipe (such as rapid interleaved transport-rendition-display). This is also due to the fact that maximum size of X11 wire packets may often be smaller than the image to be transported.

In all cases, only individual transport protocol requests are atomic. The transport of of all image data/information that makes up an IOB may or may not be atomic depending upon whether it requires multiple transport protocol requests. The image data which arrives on the server (client->server) and is placed into an IOB is not locked in any way and may be accessed by other server functions (interleaved between the transport functions). While this is the case, the actual presence of image data in the IOB (versus residing in intermediate transport buffers awaiting repacking or decompression) is indeterminate until the final flag is set, and applying server operations to the IOB may generate indeterminate results. The only acception to this is pipelines which have transport as their first element.

7.3.1.5 Synchronization

Like all other image extension protocol functions, transport protocol functions are asynchronous - the client initiates them and does not wait for their completion.

7.3.2 Set Transport Context

```
XIEP_SET_TRANSPORT

        iob_id                          XIT_IOB
        compression-param               CARD32
        compression-scheme              XIT_COMPRESSION_SCHEME
        tile_format                     BOOL
        component-space-organization    XIT_COMPONENT_SPACE_ORGANIZATION
        plane_mask                      CARD32
        plane_information               LISTofPLANE_DATA

        =>

        none

        service class value:            0
        errors:                         IDChoice, MATCH
        events:                         none

where:

        PLANE_DATA is:

            [
             pixel-stride               CARD8
             scanline-stride            CARD32
            ]
```

XIEP_SET_TRANSPORT is used to establish the necessary transport context information before image data wire transport can occur between a client buffer and a particular server IOB.

Iob_id is the id of an existing IOB (either newly created by XIEP_CREATE_RESOURCE or preexisting) which will serve as the source of or destination for image data transport. Compression_param is a compression parameter to be used by the various compression schemes. In particular, it is used to control the compression factor / fidelity tradeoff in DCT compression when reading compressed data from the server to the client. Compression_scheme is one of the valid compression schemes. In tiled transport, compression scheme must be set to PCM. In stream transport, it may be any of the compression schemes including PCM (see table in previous section and appendix). Tile_format is a boolean which is used to indicate if tile or stream transport is to occur. Component_space_organization describes the PCM format of the image data (independent of whether tile uncompressed, stream uncompressed, or stream compressed image data transport will occur).

Plane_mask identifies which planes will be transported where the definition of a data plane depends upon the component_organization. Plane_information is a list with one entry per plane. Each entry in the list contains, pixel-stride and scanline-stride. Scanline_stride is only meaningful in uncompressed stream transport and must be set equal to the number of bits per scanline plus the number of pad bits between scanlines. Scanline_stride is meaningless for tiles as tile size may be changed on a transport call basis. In tile transport, within each plane, all scanlines are bitwise abutted (no pad or alignment allowed). Multiple planes in tile transport are longword aligned. Note that the bits-per-component information used in

wire transport is communicated at IOB creation time (for client-to-server transports), and established in/derived from the IOB for server-to-client transports by use of the constraint mechanism. For more details on compression-scheme, component-space-organization, bits-per-component, pixel-stride, and scanline-stride see the appendix on image data attributes and organization.

The image data wire transport context is bound to a particular IOB for the duration of an image data wire transport session. The binding is established by XIEP_SET_TRANSPORT. The transport context information for the particular IOB is locked when the first image data wire transport function is invoked (XIEP_GET_TILE, XIEP_PUT_TILE, XIEP_GET_STREAM, XIEP_PUT_STREAM) and remains locked until the transport session involving this IOB is completed. This occurs when an image data wire transport function is invoked with "final" flag set to true or when transport is aborted (XIEP_ABORT_TRANSPORT).

Note that the transport context information pertains to image data organization only during wire transport. It has no bearing upon the server IOB image data organization as server's are free to chose their own image data organizations.

If the iob_id does not exist, an IDChoice error occurs. If the IOB's number_of_spectral_ components does not match the length of the component_information list, or the tile_format boolean is true and compression_scheme is not PCM, a MATCH error occurs.

7.3.3 Tile Data Transport

Tile data transport allows a client to transport tiles of image data between a client buffer and an existing IOB. The client controls tile size, its IOB location, and which data planes to transport. Only uncompressed data may be transported.

7.3.3.1 Client–>Server

```
XIEP_PUT_TILE

        iob_id                          XIT_IOB
        x                               CARD32
        y                               CARD32
        width                           CARD32
        height                          CARD32
        plane_mask                      CARD32
        final                           BOOL
        image_data                      XIT_IDATA

        =>

        none

        service class value:            0
        errors:                         IDChoice, MATCH, ACCESS
        events:                         none
```

XIEP_PUT_TILE transfers an uncompressed image data tile from a client buffer across the wire into a server IOB. The client specifies both the tile size and its destination location in the receiving IOB. The IOB must exist and the transport context be previously established (XIEP_SET_TRANSPORT).

IOB_id is the id of the receiving IOB. X and y specify the upper left hand corner (in IOB coordinates) of the destination location. Height and width specify the tile size. Plane_mask identifies which data planes are being sent. Final is a boolean which indicates if all IOB data (all tiles) have been sent. Image_data is the data segment to be transported from client to server.

Bitonal Monochromatic

Transport of bitonal monochromatic image data always occurs in the same fashion (Band_ Interleaved_by_Pixel, Band_Interleaved_by_Plane, and Bit_Interleaved_by_Plane are all degenerate). In this case, the client buffer is simply an array of bits, one per pixel.

Continuous Tone Monochromatic - Pixel Aggregated

Transport of continuous tone monochromatic image data can occur in two fashions: by Band_ Interleaved_by_Pixel and Band_Interleaved_by_Plane which are degenerate, or by Bit_ Interleaved_by_Plane. As there is only one spectral band, the image data organization in the client buffer for the first case is simply an array of pixels, n bits per pixel.

Continuous Tone Monochromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode transport of continuous tone monochromatic image data, the client has the option of sending a tile from one through n data planes. If more than one plane is to be transported, the planes must abut each other (long word alignment of the planes) in the client's buffer.

Continuous Tone Polychromatic - Band Interleaved by Pixel

Transport of continuous tone polychromatic image data can occur in any of the three formats: Band_Interleaved_by_Pixel, Band_Interleaved_by_Plane, and Bit_Interleaved_by_Plane. In the Band_Interleaved_by_Pixel mode the client buffer is once again simply an array of pixels, n bits per pixel, where the r-g-b bits make up the total n. No option exists regarding sending a single bit plane or spectral band.

Continuous Tone Polychromatic - Band Interleaved by Plane

In the Band_Interleaved_by_Plane mode, the client has the option of sending a tile from one through n (max 3) spectral bands. If more than one data plane (band in this case) is to be transported, they must abut each other (long word alignment of the planes) in the client's buffer.

Continuous Tone Polychromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode, the client has the option of sending one through n data planes (each data plane is now a bit plane). If more than one data plane (bit plane in this case) is to be transported, they must abut each other (long word alignment of the planes) in the client's buffer.

Specifying a non existent IOB generates an IDChoice error. Specifying an IOB which is unconstrained generates a MATCH error. A mismatch between the receiving IOB's component-space-organization and the current transport context's component-space-organization (ie. trying to send an rgb color image over the wire into a monochrome IOB) generates a MATCH

error. Attempts to transport to an IOB which has not had a transport context established (XIEP_SET_TRANSPORT) generates an ACCESS error.

7.3.3.2 Server–>Client

```
XIEP_GET_TILE

        iob_id                          XIT_IOB
        x                               CARD32
        y                               CARD32
        width                           CARD32
        height                          CARD32
        plane_mask                      CARD32
        final                           BOOL

        =>

        image_data                      XIT_IDATA

        service class value:            0
        errors:                         IDChoice, MATCH, ACCESS
        events:                         none
```

XIEP_GET_TILE transfers an uncompressed image data tile from a server IOB across the wire into a client buffer. The client specifies both the tile size and its source location in the sending IOB. The IOB must be preexisting, and the transport context previously established (XIEP_SET_TRANSPORT).

IOB_id is the id of the sending IOB. X and y specify the upper left hand corner (in IOB coordinates) of the sending location. Height and width specify the tile size. Plane_mask identifies which data planes are being requested. Final is a boolean which indicates if all IOB data (the final tile) has been asked for. Image_data is the data segment received from the server.

Bitonal Monochromatic

Transport of bitonal monochromatic image data always occurs in the same fashion (Band_ Interleaved_by_Pixel, Band_Interleaved_by_Plane, and Bit_Interleaved_by_Plane are all degenerate). In this case, the client buffer will be an array of bits, one per pixel.

Continuous Tone Monochromatic - Aggregated

Transport of continuous tone monochromatic image data can occur in two fashions: by Band_ Interleaved_by_Pixel and Band_Interleaved_by_Plane which are degenerate, or by Bit_ Interleaved_by_Plane. As there is only one spectral band, the image data organization in the client buffer for the first case is simply an array of pixels, n bits per pixel.

Continuous Tone Monochromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode transport of continuous tone monochromatic image data, the client has the option of receiving a tile from one through n data planes. If more than one plane is to be transported, the planes must abut (long word alignment of planes) in the client's buffer.

Continuous Tone Polychromatic - Band Interleaved by Pixel

Transport of continuous tone polychromatic image data can occur in any of the three formats: Band_Interleaved_by_Pixel, Band_Interleaved_by_Plane, and Bit_Interleaved_by_Plane. In the Band_Interleaved_by_Pixel mode the client buffer is once again simply an array of pixels, n bits per pixel, where the r-g-b bits make up the total n. No option exists regarding receiving a single bit plane or spectral band.

Continuous Tone Polychromatic - Band Interleaved by Plane

In the Band_Interleaved_by_Plane mode, the client has the option of receiving a tile from one through n (max 3) spectral bands. If more than one data plane (band in this case) is to be transported, they will abut each other (long word alignment of the planes) in the client's buffer.

Continuous Tone Polychromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode, the client has the option of receiving one through n data planes (each data plane is now a bit plane). If more than one data plane (bit plane in this case) is to be transported, they will abut each other (long word alignment of the planes) in the client's buffer.

Specifying a non existent IOB generates an IDChoice error. Specifying an IOB which is unconstrained generates a MATCH error. A mismatch between the sending IOB's component-space-organization and the current transport context's component-space-organization generates a MATCH error. Attempts to transport from an IOB which has not had a transport context established (XIEP_SET_TRANSPORT) generates an ACCESS error.

7.3.4 Stream Data Transport

Stream data transport allows a client to stream image data or image code information (compressed image data) between a client buffer and a new (PUT) or existing (GET) IOB. In the case of image code information (compressed image data), the server compress/decompress step is hidden (happens without client intervention). The client controls both the compression format and the length of stream data. Unlike tile transport, only one data plane may be transported with each stream transport protocol request. Note that this does not prevent clients from interleaving stream transport of different planes. For stream transport, the data can only be sent/received in sequential (raster) order - no random access is possible.

Note that unlike tile transport, in stream transport image data or image code information is being sent sequentially, and not image data tiles randomly. This data or information is in essence aggregated sequentially into one or more buffers and then, if necessary, decompressed to produce the image data. Particulars of decompression on the server side of the wire depend upon both the compression scheme involved and the server implementation (ie. G4 decompression may be conducted as code information arrives as may DCT decompression, or the server may chose to wait for the complete code information to be available before decompression).

7.3.4.1 Client–>Server

```
XIEP_PUT_STREAM

        iob_id                          XIT_IOB
        byte_count                      CARD32
        final                           BOOL
        plane_num                       CARD8
        image_data                      XIT_IDATA

        =>

        none

        service class value:            0
        errors:                         IDChoice, MATCH, ACCESS
        events:                         none
```

XIEP_PUT_STREAM transfers image data or image code information (compressed image data) from a client buffer across the wire into a server IOB. If image code information is transported, it is decompressed into image data before being placed in the IOB. The IOB must be newly created (XIEP_CREATE_RESOURCE), and the transport context previously established (XIEP_SET_TRANSPORT).

IOB_id is the id of the receiving IOB. Byte_count specifies the number of bits of valid image data or code information in the buffer. Final is a boolean which indicates if all image data or code information has been sent. Plane_num identifies which bit plane is being sent. Image_ data is the image data or image code information (compressed image data) to be transported from client to server.

Bitonal Monochromatic

Transport of bitonal monochromatic image data or image code information (compressed image data) always occurs in the same fashion (Band_Interleaved_by_Pixel, Band_Interleaved_by_ Plane, and Bit_Interleaved_by_Plane are all degenerate). In the stream uncompressed (PCM) mode, the client buffer is simply an array of butted data bits (1 bit per pixel). In the stream compressed mode (G4) the client buffer is an array of butted code bit strings.

Continuous Tone Monochromatic - Pixel Aggregated

Transport of continuous tone monochromatic image data or image code information (compressed image data) can occur in two fashions: by Band_Interleaved_by_Pixel and Band_ Interleaved_by_Plane which are degenerate, or by Bit_Interleaved_by_Plane. In the first case, for stream uncompressed (PCM) mode, the client buffer is simply an array of butted data bits (n bits per pixel). In the stream compressed mode (DCT) the client buffer is an array of butted code bit strings.

Continuous Tone Monochromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode transport of continuous tone monochromatic image data or image code information (compressed image data), the client may only send image data or image code information (compressed image data) from one data plane (bit plane) in each transport call. In the stream uncompressed (PCM) mode, the bit plane in the client's buffer is simply an array of butted data bits. In the stream compressed mode (G4) the client buffer is an array of butted code bit strings.

Continuous Tone Polychromatic

Transport of continuous tone polychromatic image data or image code information (compressed image data) can occur in three formats: Band_Interleaved_by_Plane, Bit_Interleaved_by_ Plane, and Band_Interleaved_by_Pixel.

Continuous Tone Polychromatic - Band Interleaved by Pixel

In the Band_Interleaved_by_Pixel mode, uncompressed image data or image code information (compressed image data) for all bands is sent.

Continuous Tone Polychromatic - Band Interleaved by Plane

In the Band_Interleaved_by_Plane mode, the client has the option of sending image data or image code information (compressed image data) for any one plane (spectral band) In the stream uncompressed (PCM) mode, the band in the client's buffer is simply an array of butted data bits. In the stream compressed mode (DCT) the client buffer is an array of butted code bit strings for the particular band.

Continuous Tone Polychromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode, the client has the option of sending image data or image code information (compressed image data) from any one data plane (bit plane). In the stream uncompressed (PCM) mode, the bit plane in the client's buffer is simply an array of butted data bits (1 per pixel). In the stream compressed mode (G4) the client buffer is an array of butted code bit strings.

Specifying a non existent IOB generates an IDChoice error. Specifying an IOB which is unconstrained generates a MATCH error. A mismatch between the receiving IOB's component-space-organization and the current transport context's component-space-organization (ie. trying to send an rgb color image over the wire into a monochrome IOB) generates a MATCH error. Attempts to transport from an IOB which has never had a transport context established (XIEP_SET_TRANSPORT) generates an ACCESS error.

7.3.4.2 Server–>Client

```
XIEP_GET_STREAM

        iob_id                          XIT_IOB
        max_bytes                       CARD32
        plane_num                       CARD8

        =>

        final                           BOOL
        byte_count                      CARD32
        image_data                      XIT_IDATA

        service class value:            0
        errors:                         IDChoice, MATCH, ACCESS
        events:                         none
```

XIEP_GET_STREAM transfers image data or image code information (compressed image data) from a server IOB across the wire to a client buffer. In compressed mode, the server extracts image data from the IOB and compresses it before putting it on the wire. The IOB must be preexisting, and the transport context previously established (XIEP_SET_TRANSPORT).

IOB_id is the id of the sending IOB. Max_bytes is the maximum number of bytes the client is prepared to receive (typically buffer size). Plane_num identifies which bit plane is to be sent.

Final is a boolean which indicates if all image data or code information has been sent. Byte_ count specifies the number of bytes of valid image data or code information in the buffer. Image_data is the image data or image code information (compressed image data) which has bene transported from server to client.

Bitonal Monochromatic

Transport of bitonal monochromatic image data or image code information (compressed image data) always occurs in the same fashion (Band_Interleaved_by_Pixel, Band_Interleaved_by_ Plane, and Bit_Interleaved_by_Plane are all degenerate). In the stream uncompressed (PCM) mode, the client buffer will contain an array of butted data bits (1 bit per pixel). In the stream compressed mode (G4) the client buffer will contain an array of butted code bit strings.

Continuous Tone Monochromatic - Pixel Aggregated

Transport of continuous tone monochromatic image data or image code information (compressed image data) can occur in two fashions: by Band_Interleaved_by_Pixel and Band_ Interleaved_by_Plane which are degenerate), or by Bit_Interleaved_by_Plane. In the first case, in the stream uncompressed (PCM) mode, the client buffer will be an array of butted data bits (n bits per pixel). In the stream compressed mode (DCT) the client buffer will be an array of butted code bit strings.

Continuous Tone Monochromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode transport of continuous tone monochromatic image data or image code information (compressed image data), the client has the option of receiving the image data or image code information (compressed image data) from one bit plane. In the stream uncompressed (PCM) mode, each bit plane in the client's buffer will be an array of butted data bits. In the stream compressed mode (G4) the client buffer will be an array of butted code bit strings.

Continuous Tone Polychromatic

Transport of continuous tone polychromatic image data or image code information (compressed image data) can occur in three formats: Band_Interleaved_by_Plane, Bit_Interleaved_by_ Plane, and Band_Interleaved_by_Pixel.

Continuous Tone Polychromatic - Band Interleaved by Pixel

In the Band_Interleaved_by_Pixel mode, uncompressed image data or image code information (compressed image data) for all bands is sent.

Continuous Tone Polychromatic - Band Interleaved by Plane

In the Band_Interleaved_by_Plane mode, the client has the option of receiving image data or image code information (compressed image data) from one spectral band (all_planes false). In the stream uncompressed (PCM) mode, each band in the client's buffer is simply an array of butted data bits. In the stream compressed mode (DCT) the client buffer will be an array of butted code bit strings.

Continuous Tone Polychromatic - Bit Interleaved by Plane

In the Bit_Interleaved_by_Plane mode, the client has the option of receiving image data or image code information (compressed image data) from one bit plane. In the stream uncompressed (PCM) mode, each bit plane in the client's buffer will be an array of butted data bits (1 per pixel). In the stream compressed mode (G4) the client buffer will be an array of butted code bit strings.

Specifying a non existent IOB generates an IDChoice error. Specifying an IOB which is unconstrained generates a MATCH error. A mismatch between the receiving IOB's component-space-organization and the current transport context's component-space-organization (ie. trying to send an rgb color image over the wire into a monochrome IOB) generates a MATCH error. Attempts to transport from an IOB which has never had a transport context established (XIEP_SET_TRANSPORT) generates an ACCESS error.

7.3.5 Abort Transport

```
XIEP_ABORT_TRANSPORT

        iob_id                          XIT_IOB
        plane_mask                      CARD32

        =>

        none

        service class value:            0
        errors:                         none
        events:                         none
```

XIEP_ABORT_TRANSPORT is used to abort wire transport before a final flag is sent.

IOB_ID is the id of the IOB which is the source of (server->client) or the destination for (client->server) image data and to/from which image data is being transported. Plane_mask is a bit mask with one bit per image plane. Image transport may be aborted on a per plane basis. When the server receives an abort request, all image data which is in intermediate buffers (not yet sent, or not yet overlaid onto the IOB) is deleted, and the final flag is set for that plane.

There is no return value from XIEP_ABORT_TRANSPORT, and no possible errors.

7.4 Process Services

Process services provide for the application of low level image operators to image data primarily in support of preparation for or improvement of its display. These functions take one or more input operands including existing image objects, and produce a two dimensional array of data which can be treated as either a new (modified or enhanced) image object, or a control plane for other operations. As such, process services can be consider as "image in –> image out" functions.

Many process protocol functions take a constant as one of the image operands. This constant is a shorthand format for an image object of constant value. The server in essence constructs a "logical" image object whose value is equal to the specified constant at all locations, and whose extent is sufficient for the operation at hand (typically identical in size to the other "physical" image operand). These "constant image objects" are temporary entities which exist only during the operation and can not be referenced by the client outside of the operation. "Constant" images are cast by the server as type constrained or unconstrained as the situation demands.

All process services take an image domain control context (either a region of interest {ROI} or a control processing plane {CPP}) and operate on the pixels identified by the domain control context. The resulting image object is identical in size to either the subset of the input image defined by an ROI type IDC, or to the original image object if a CPP type IDC is used. In the latter case, the output image data is valid only at the points specified by the CPP. Data at all other points is unpredictable.

Constrained and unconstrained model operation is particularly relevant to process services. Each subsection of the process services section contains a paragraph which outlines constrained and unconstrained modes of operation. Individual functions note exceptions to the overall comments. Where a function can only accept and produce constrained image data, this is noted and no "constrained" control boolean is provided. Where the option does exist, the protocol function includes a "constrained" control boolean which informs the server to apply the default constraint model and produce constrained output.

Process services can be applied to mono and polychromatic (rgb) images. In general, the functions will accept lists of parameters (one entry per spectral component), and will operate on each of the three spectral components (r, g, b) as if they were three separate IOB's. Different parameter values per spectral component are possible . Clients who wish to operate only on a single spectral component may be able to set parameter values in certain cases to make this happen. If not, clients have the option of expanding the spectral components into separate images, operating individually on the components, and then recombining them. Exceptions are noted on a function by function basis.

While control over protocol implementation (server design) is out of the realm of this specification, its is the author's intent to limit processing to spatial domain techniques (vs spatial frequency) at this time. As such, the protocol does not support the definition or communication of spatial frequency domain (such as Fourier) data (including images and filters), or spatial frequency domain processes.

Processing services are segmented into the following categories and include the following functions:

- Point Processing

- Area Processing
- Ensemble Processing
- Geometric Processing
- Radiometric Processing

7.4.1 Point Processing

Point processing allows a client to apply a client specified transfer function to remap an input pixel value to an output pixel value with no dependency upon surrounding (area) input pixels. Point processing can decrease, preserve, or increase the number of bits per pixel per component (gray->b&w, b&w->b&w, b&w->gray) and preserve or increase the number of spectral components (mono->mono, color->color, mono->color). Typically, it is used for gray scale luminance transforms (gray->gray), simple color remapping (color->color), and pseudo coloring (b&w->color, gray->color). While a color image can be remapped into another color image with equal r-g-b values on a per pixel basis and result in a "gray scale" display, the underlying data remains multispectral.

```
XIEP_POINT

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        idc                             XIT_IDC
        new_quantum                     ListofCARD8
        transfer_function               ListofListofCARDnn

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, VALUE,MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_POINT creates a new server IOB from an existing server IOB by remapping each input IOB value into a new output IOB value. The output value depends solely upon the input pixel value (no source area dependency). The function is designed to allow the client to apply a client specified transfer function.

IOB_id1 is the id of the source image object. IOB_id2 is the client specified id to be assigned to the newly created (resulting) image object. Idc controls the subset of the image which will be operated upon. Transfer_function is a list of what are in essence look up tables - one per output spectral component. Each one must be long enough so that all potential values of the input can be used as an index into the table. New_quantum is a list (one per spectral component) of the number of bits per pixel (per spectral component) of the resulting value which derives from the look up table. This allows the transfer function to both modify image data values, and bits per pixel per spectral component. If the input IOB is monochromatic and the client specifies one transfer_function, the output will also be monochromatic. If the input is monochromatic and the user specifies 3 transfer_functions, the output will be trichromatic.

The input IOB must be constrained, and the output IOB is constrained. The input image object may be either mono or poly chromatic. The output image object will at least be mono chromatic, but may be promoted to poly chromatic as discussed above.

XIEP_POINT operates on the subset of the input image defined by the IDC. If the IDC is of type CPP, the output image object is equal in size to that of the input image object, but valid only at those points defined by the CPP. If IDC is of type ROI, the output image object is less than or equal to the size of the input image object.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. An out of range new quantum value generates a VALUE error. Specification of an rgb input IOB and a single transfer_function results in a MATCH error as does specification of an unconstrained input IOB.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.2 Area Processing

Area processing produces a data response at each pixel based upon area operations at and around the source pixel. The data response array may be considered to be an image and is stored as an IOB.

Area processing takes a single input image object operand, a single input IDC operand of type ROI or CPP, a single input IDC operand of type TMP, and produce a single image object output operand. The value of the output image object at each point [selected by the IDC of type ROI or CPP], is determined in a similar fashion for each areawise operation:

1. The pointwise arithmetic combination (sum, product) of the TMP coefficient and the source image operand are taken (this produces a 2d array of coefficients some of which may intentionally be undefined - ie. potential for small size, and / or sparce population).

2. An algebraic operator is applied to this data (sum, minimum, maximum) which reduces the two dimensional array to a single value (addition adds up all of these values, minimum and maximum select the minimum and maximum values respectively.

Note that the most familiar of these area operations is implemented as a pointwise (across the area) product, followed by an area summation. This is the normal convolution process.

The location (overlay or registration) of the template relative to the image point in question is established by the template's "centering" parameter (see XIEP_CREATE_IDC description of templates).

```
XIEP_AREA

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        idc_c                           XIT_IDC {ROI,CPP}
        idc_t                           XIT_IDC {TMP}
        operation1                      XIT_AREA_OP1
        operation2                      XIT_AREA_OP2
        constrain                       BOOL

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_AREA performs generalized area functions including additive and multiplicative, minimization, maximization, and summation. In particular, it adds or multiplies the TMP contents to the image contents on a pointwise basis and then minimizes, maximizes (selects the minimum or maximum), or sums the results.

IOB_id1 is the source image object. IDC_c (ROI, CPP) controls the subregion to which the overall process is applied. IDC_t (TMP) contains the area coefficients and controls the subset of the source image surrounding the current pixel to which the process applies. IOB_id2 is the client specified id to be assigned to the newly created (resulting) image object. Constrain determines if the default constraint model will be applied upon completion.

In the case where operation1 is multiplication and operation2 is summation, XIEP_AREA performs generalized convolution. In particular, it convolves a user specified convolution kernel (represented by the TMP type IDC) with the subset of the source image object producing a new image object. IDC_t (TMP) contains the kernel coefficients and controls the subset of the image to which the summation applies.

Note that by using the template mechanism (IDC of type TMP) both fully and partially populated convolution kernels may be represented and applied. The convolution kernel must be smaller in extent than the region of the image to which it will be applied. Practically, 3 x 3 and 5 x 5 convolution kernels should be considered. The process consists of multiplying the TMP contents by the image contents on a pointwise basis, and summing the results.

The source image object must be large enough to encompass all the points identified by the IDC.

The input image object maybe either constrained or unconstrained. If the source image is unconstrained or the process produces a value outside of the dynamic range of the quantized component, conversion to unconstrained occurs. If the constrain boolean is true, the constraint model is applied producing constrained output data.

The input image object may be either mono or poly chromatic. The output image object will be of the same type (mono or poly chromatic). As the TMP type IDC has no provision for specification of parameters by spectral component, the same template parameters and operation will be applied to all spectral components. Clients desiring different TMP's per spectral component should separate the spectral components, operate upon them separately, and then recombine them.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.3 Ensemble Processing

In ensemble processing, operations are performed on pointwise basis between spatially registered points in the two image operands. The IDC is used simply to identify connected or disconnected subregions to be used as input operand data (IDC is simply a boolean selector).

Ensemble processing is divided into four classes:

- Arithmetic
- Relational
- Logical
- Mathematical

All classes can accept and potentially produce unconstrained image data except the logical class which will only accept and produce constrained image data. For all other classes, where multiple images are input to the operation, if all image inputs are not the same (either constrained or unconstrained), constrained data is promoted to unconstrained before the operation is conducted.

In general, all ensemble services operate on either mono or poly chromatic images. For poly chromatic images, the operations are performed on a band by band basis and the resulting image is also poly chromatic.

7.4.3.1 Arithmetic

```
XIEP_ARITH

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        constant                        LISTofCARDnn
        iob_id3                         XIT_IOB
        idc                             XIT_IDC
        operator                        XIT_ARITH_OP
        constrain                       BOOL
        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH, VALUE
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_ARITH performs arithmetic functions (add, subtract, multiply, divide, minimum, maximum) between two image objects or between an image object and a constant (image) producing a new image object.

IOB_id1 and iob_id2 are the two source image objects, and constant is the constant image. When two IOB's are involved, iob_id2 is non null and constant is ignored. If the operation involves a constant, iob_id2 is null and constant is non null. IOB_id3 is the client specified id to be assigned to the newly created (resulting) image object. IDC controls the subset of the images which will be operated upon. Operator controls which arithmetic operation will be performed between the two images or between the image and the constant (image). Constrain determines if the default constraint model will be applied upon completion.

The source image objects need not be the same size, but they must be large enough to encompass all the points identified by the IDC. As the IDC is IOB relative, it is the users responsibility to ensure spatial alignment of the two IOB contents.

The input image objects maybe either constrained or unconstrained. If the operation involves a constant image and an unconstrained IOB, the constant will be promoted to unconstrained before the operation is performed. If either image is unconstrained or the operation produces a value outside of the dynamic range of the quantized component, conversion to unconstrained occurs. Thus, the resulting image object may be constrained or unconstrained. If the constrain boolean is true, the default constraint model will be applied upon completion.

The source images must have the same number of spectral components or an error is generated. Arithmetic operations occurs on a spectral component by spectral component basis.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. Difference between source IOB's number of spectral components generates a MATCH error. Arithmetic failures (overflow, underflow, divide by zero) generate a VALUE error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.3.2 Relational

```
XIEP_COMPARE

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        constant                        LISTofCARDnn
        cpp                             LISTofXIT_IDC {CPP}
        idc                             XIT_IDC {CPP or ROI}
        operator                        XIT_PRED_TYPE
        combine                         BOOL

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_COMPARE compares two image objects or an image object and a constant on a pointwise basis and generates a CPP type IDC (boolean bitmap) which reflects the results of the pointwise comparison. It may be used to perform either a spectral component by spectral component comparison or a complete image (all components taken together) comparison.

IOB_id1 and iob_id2 are the two source image objects, and constant is the constant value. When two IOB's are involved, iob_id2 is non null and constant is ignored. If the operation involves a constant, iob_id2 is null and constant is non null. LISTofCpp is the client assigned id's for the newly created control processing planes which contain the comparative results. Idc controls the subset of the images which will be compared. Operator is the logical predicate operator used in the comparison. Combine is a boolean which determines if the comparison should be done on a spectral component by spectral component basis, or on an all components basis.

The source image objects need not be the same size, but they must be large enough to encompass all the points identified by the IDC. As the IDC is IOB relative, it is the users responsibility to ensure spatial alignment of the two IOB contents.

The input image objects maybe either constrained or unconstrained. If the operation involves a constant image and an unconstrained IOB, the constant will be promoted to unconstrained before the operation is performed. If the input image objects are not both constrained, both are converted to unconstrained before comparison occurs. The source images must have the same number of spectral components.

If the source images are monochromatic, there is only one output CPP. If the source images are polychromatic and the combine boolean is true, there is still only one output CPP. If the source images are polychromatic and the combine boolean is false, there are three output CPP's.

The combine option is only meaningful for the "equal to" and "not equal" to operators. In these cases, equality is established on a component by component basis and then 'anded' to derive image equality. Inequality is then simply a not of this result. These combine operations are provided as a convenience packaging (they can easily be built using other protocol functions).

The function does not support rgb distance metrics, and so, combined less than, less than or equal to, greater than, greater than or equal to function are not supported. Clients may implement these by using the spectral separate, mathematic process, arithmetic process, and relational process functions.

The output CPP has a value of 1 where ever the expression is true, and a value of 0 where it is false or the controlling IDC prevents comparison.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.3.3 Logical

```
XIEP_LOGICAL

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        constant                        LISTofCARDnn
        iob_id3                         XIT_IOB
        idc                             XIT_IDC
        operator                        XIT_LOGICAL_OPS

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_LOGICAL performs bitwise (logical plane wise) operations on individual images, and between two IOB's or an IOB and constant (image). All logical operators are pointwise, and all produce a new image object as a result.

IOB_id1 and iob_id2 are the two source image objects, and constant is the constant value. When two IOB's are involved, iob_id2 is non null and constant is ignored. If the operation involves a constant (image or otherwise), iob_id2 is null and constant is non null. IOB_id3 is the client specified id to be assigned to the newly created (resulting) image object. Idc controls the subset of the images which will be logically combined. Operator is the logical combination operator to be used.

In the case of the 'not' operation, bitwise inversion is performed. In the case of 'shift_up' and 'shift_dn', the constant determines how many bits. Bits shifted in are zero filled.

The source images must have the same number of spectral components. Combining (and, or, xor) occurs a spectral component by spectral component basis. and on a bit wise basis within spectral component. If the images differ in depth, the shallower is extended to the same depth as the deeper, with all additional planes being zero filled. The resulting image has the same number of spectral components as the source images.

The source image objects need not be the same size, but they must be large enough to encompass all the points identified by the IDC. As the IDC is IOB relative, it is the users responsibility to ensure spatial alignment of the two IOB contents.

The input image objects may only be constrained. The resulting image object is also constrained.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. Specification of an unconstrained IOB or spectral mismatches between IOB's generates a MATCH error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.3.4 Mathematical

```
XIEP_MATHF

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        idc                             XIT_IDC
        function                        XIT_MATHF
        constrain                       BOOL

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, VALUE
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_MATHF applies a single operand mathematical function to the input image on a pointwise basis (at all points selected by the idc) producing a new image.

IOB_id1 is the source image object. IDC controls the subset of the image to which the math function will be applied. IOB_id2 is the client specified id to be assigned to the newly created (resulting) image object. Function selects which mathematical function will be applied. Constrain determines if the default constraint model will be applied upon completion.

The source image object must be large enough to encompass all the points identified by the IDC.

While the input image object maybe either constrained or unconstrained, in most cases, the output object will be unconstrained. If the constrain boolean is true, the default constraint model will be applied upon completion.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. Mathematical errors such as overflow and invalid input (sqrt(-1)) generate a VALUE error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.4 Geometric

Geometric services provide for the ability to choose a subset of an image for display, and adjust the image with regards to the two spatial dimensions. Subsetting is provided for by a crop function, adjustment in the two spatial dimensions via translate, scale, rotate, and mirror functions.

All geometric services operate on either mono or poly chromatic images. For poly chromatic images, the operations are performed individually on each of the bands and the resulting image is also poly chromatic.

Geometric protocol functions include:

- XIEP_SCALE - scale image in x and y directions
- XIEP_ROTATE - rotate image around z axis
- XIEP_MIRROR - rotate image around x and/or y axes
- XIEP_CROP - extract a subset of an image
- XIEP_TRANSLATE - shift and combine two images

All geometric services can accept either constrained or unconstrained image data in. No geometric services offer a constrain control boolean. The output image will be of the same type (constrained or unconstrained) as the input.

7.4.4.1 Crop Image

```
XIEP_CROP

        src_iob                 XIT_IOB
        dst_iob                 XIT_IOB
        idc                     XIT_IDC {ROI}

        =>

        none

    service class value:         0
    errors:                      IDChoice, ALLOC, VALUE
    events:                      XIE_COMPUTATION_COMPLETE
```

XIEP_CROP creates a new server IOB from an existing server IOB by cropping the old IOB data in either or both the X and Y dimensions.

SRC_iob is the id of the source image object. DST_iob is the id of the destination IOB (to be created). The idc of type roi defines the geometric extent of the source image to be cropped. The server creates a new IOB to receive the cropped image data and attributes, and uses the client specified dst_iob as its id.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. An out of range parameter value generates a VALUE error.

An XIE_COMPUTATION_COMPLETE event occurs when the destination IOB is created and filled with the resulting cropped data.

7.4.4.2 Translate Image

```
XIEP_TRANSLATE

        src_iob                         XIT_IOB
        dst_iob                         XIT_IOB
        src_idc                         XIT_IDC
        dst_idc                         XIT_IDC
        constant                        LISTofCARDnn

        =>

        none

    service class value:        0
    errors:                     IDChoice, ALLOC, VALUE, MATCH
    events:                     XIE_COMPUTATION_COMPLETE
```

XIEP_TRANSLATE allows one image to be translated or shifted relative to another image and then combined via simple overlay. In essence, it is the inverse of crop as it provides for a simple paste operation.

SRC_IOB acts as the source of image data for the translate and combine operation. DST_IOB acts as the destination or receiving location of the translate and combine operation. SRC_IDC determines the subset of SRC_IOB that will be used. DST_IDC determines the location and clip boundaries in DST_IOB that will receive the source image data.

The two IDC's (SRC_IDC and DST_IDC) are conceptually aligned at their upper left corner. If SRC_IDC is smaller than DST_IDC in either direction, the remaining pixels in the area prescribed by the DST_IDC are filled with the CONSTANT value in that direction. If SRC_ IDC is larger than DST_IDC in either direction, the source data is clipped to fit with DST_IDC in that direction.

Note that both SRC_IOB and SRC_IDC are optional (both must be present or absent). If they are absent, CONSTANT is used as the source data which is pasted into the destination image area prescribed by DST_IDC.

The server creates a new IOB to receive the combined image data and attributes, and uses the client specified dst_iob as its id.

Note that this function only supports replacement or non transparent overlay (combination) and not alpha plane control mixing or translucency.

Both images must be constrained or unconstrained and mono chromatic or polychromatic. If the operation involves a constant image and an unconstrained IOB, the constant will be promoted to unconstrained before the operation is performed.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. An out of range parameter value generates a VALUE error. Mismatch in constraints or chromaticity between source images generates a MATCH error.

An XIE_COMPUTATION_COMPLETE event occurs when the destination IOB is created and filled with the resulting cropped data.

7.4.4.3 Scale Image

```
XIEP_SCALE

        src_iob                 XIT_IOB
        dst_iob                 XIT_IOB
        pixels_per_scanline     CARD32
        number_of_scanlines     CARD32

        =>

        none

    service class value:        0
    errors:                     IDChoice, ALLOC, VALUE
    events:                     XIE_COMPUTATION_COMPLETE
```

XIEP_SCALE creates a new server IOB from an existing server IOB by scaling the old IOB data in either or both the x and y dimensions.

SRC_iob is the id of the source image object. DST_iob is the id of the destination IOB (to be created). Pixels_per_scanline is the number of pixels per scan line for the scaled image. Number_of_scanlines is the number of scan lines in the scaled image.

The two IOB's may differ in either or both the number of pixels per scan line and the number of scan lines. The ratio of new to old values of these two sets may be considered to be the x and y scale factors. Scaling up or down (new value leq or new value geq old value) is allowed. Scaling in one or both dimensions at the same time is allowed. Non integral ratios (scale factors) is allowed.

XIEP_SCALE_IMAGE operates on the complete input image to produce the scaled output image. Those attempting to scale a portion of a source image should use crop to extract the sub image into a new IOB, and then scale it.

With knowledge of the display device characteristics and the original source object, a client can achieve combinations of the following:

- scale to fit within a window
- scale preserving original aspect ratio
- scale to reproduce at original size
- ...

The server creates a new IOB to receive the scaled image data and attributes, and uses the client specified dst_iob as its id.

Servers are free to choose their own scaling algorithms (including separate integral and non integral ones).

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. An out of range parameter value generates a VALUE error.

An XIE_COMPUTATION_COMPLETE event occurs when the destination IOB is created and filled with the resulting scaled data.

7.4.4.4 Rotate Image

```
XIEP_ROTATE

        src_iob                 XIT_IOB
        dst_iob                 XIT_IOB
        angle                   XIT_FLOAT
        width                   CARD32
        height                  CARD32
        fill                    LISTofCARD32

        =>

        none

    service class value:        0
    errors:                     IDChoice, ALLOC, VALUE
    events:                     XIE_COMPUTATION_COMPLETE
```

XIEP_ROTATE creates a new server IOB from an existing server IOB by rotating the source IOB data around the z axis (normal to the 2d image data x-y plane) through the center of the source IOB.

SRC_iob is the id of the source image object. DST_iob is the id of the destination IOB (to be created). Angle is the rotation angle in degrees measured clockwise around the z axis. Width and height are parameters which determine the resulting size of the rotated image (dst_iob). Fill is the value of a pixel to use in areas where the pixel value is undetermined by rotation.

XIEP_ROTATE operates on the complete input image to produce the rotated output image. Those attempting to rotate a portion of a source image should use crop to extract the sub image into a new IOB, and then rotate it.

The width and height parameters operate independently, but in a similar fashion. They are used to determine the size of the resulting image (dst_iob). If either or both is zero, the algorithm determines the smallest bounding dimension in that direction which will contain all of the rotated source data, and sets the resulting image object to that size. In essence, setting width and/or height to zero ensures that no rotated data is clipped in the horizontal and/or vertical direction, and that a minimal bounding size is chosen.

If either or both is nonzero, the algorithm will set the resulting image object to the specified size. If the rotated data is larger in this direction than the destination image object, the image is clipped symmetricaly.

In either case, the rotated data will be centered in the resulting image object, and pixels in the rotated IOB which do not derive from source IOB pixels will have their value set to that specified in the Fill parameter. Centering the image (due to rotation around the center of the source image) results in symmetric clipping when clipping occurs.

Servers are free to choose their own rotation algorithms (including separate integral and non integral ones).

Invalid src_iob results in an IOB error. Inability to allocate the rotated IOB results in an ALLOC error, while invalid parameter values result in VALUE errors.

An XIE_COMPUTATION_COMPLETE event occurs when the destination IOB is created and filled with the resulting rotated data.

7.4.4.5 Mirror Image

```
XIEP_MIRROR

        src_iob                 XIT_IOB
        dst_iob                 XIT_IOB
        x-mirror                BOOL
        y-mirror                BOOL

        =>

        none

    service class value:        0
    errors:                     IDChoice, ALLOC
    events:                     XIE_COMPUTATION_COMPLETE
```

XIEP_MIRROR creates a new server IOB from an existing server IOB by rotating the old IOB data around the x axis, y axis, or x & y axes of the source IOB. Only rotations of 180 degrees are allowed which is why the function is named Mirror. This is in keeping with the package supporting only 2d image data (no perspective transforms allowed or meaningful).

SRC_iob is the id of the source image object. DST_iob is the id of the destination IOB (to be created). X-mirror and Y-mirror are booleans which determines if the mirror should occur around the source IOB's x, y, or x & y axes.

XIEP_MIRROR operates on the complete input image to produce the mirrored output image. Those attempting to mirror a portion of a source image should use crop to extract the sub image into a new IOB, and then mirror it.

The server creates a new IOB to receive the mirrored image data and attributes, and uses the client specified dst_iob as its id.

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error.

An XIE_COMPUTATION_COMPLETE event occurs when the destination IOB is created and filled with the resulting mirrored data.

7.4.5 Radiometric

Radiometric services facilitate the adjustment of an image with regards to the luminance and/or chrominance dimensions. Radiometric functions are used to directly affect these distributions or prepare data for adjustment through the use of other point, area, and ensemble processing services.

Radiometric protocol functions include:

- XIEP_CONSTRAIN - apply a quantization model
- XIEP_MATCH_HISTOGRAM - transform the image to have the specified histogram form
- XIEP_CHROMINANCE_SEPARATE - separate a poly chromatic image into multiple monochromatic images
- XIEP_CHROMINANCE_COMBINE - combine multiple mono chromatic images into a poly chromatic image
- XIEP_DITHER - reduce number of quantization levels through an area technique
- XIEP_LUMINANCE - extract the luminance component from a poly chromatic image

7.4.5.1 Constrain

```
XIEP_CONSTRAIN
        iob_id1                         XIT_IOB
        =>
        none
    service class value:                1
    errors:                             IDChoice
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_CONSTRAIN is used to convert an unconstrained IOB into a constrained IOB. In essence, it applies a client selected quantization model to the image data. Application of the quantization model may involve steps such as absolute value, range shifting and scaling, hard clipping, and mapping to a fixed number of quantization levels. While it is possible to perform some of these functions through other protocol primitives, XIEP_CONSTRAIN packages them in an essential way which support the transition from unconstrained data generated by some server functions to constrained data that can move towards display. As such, it is considered a primitive function.

IOB_id1 is the image object whose state is to be changed from unconstrained to constrained. The constraint model applied is the current default constraint model. If the input image is already constrained, and the current default constraint model is the same as the one last applied to the image, then the function is a noop. If not, the current default constraint model will be applied.

Constrain will operate on either mono chromatic or poly chromatic images.

No subseting by IDC mechanism is possible (the entire image is constrained).

Specification of an invalid source IOB, or an already in use resource id for the destination IOB generates an IDChoice error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.5.2 Histogram Matching

```
XIEP_MATCH_HISTOGRAM

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        idc                             XIT_IDC
        h_shape                         XIT_HSHAPE
        parm1                           XIT_FLOAT
        parm2                           XIT_FLOAT

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_MATCH_HISTOGRAM creates a new server IOB from an existing server IOB where the output IOB differs from the input in terms of its component distribution or histogram. The function is designed to allow the client to specify the desired state of the resulting image's histogram (algorithm description of resulting histogram shape). It implements operations such as histogram equalization (flattening) and hyperbolization.

IOB_id1 is the id of the source image object. IDC_id controls the subset of the image which will be operated upon. H_shape is the user specified form of the desired output image histogram. IOB_id2 is the client specified id to be assigned to the newly created (resulting) image object. Parm1 and 2 are parameters which are used to adjust the algorithmic shapes. They will be specified on an hshape by hshape basis (tbs).

The input IOB must be constrained and monochromatic. Output IOB's are always constrained and monochromatic.

XIEP_MATCH_HISTOGRAM operates on the subset of the input image defined by the IDC. Only IDC's of type ROI are supported. Histogram matching is implemented on the server side of the wire because it can not be implemented by a simple transfer function. In particular, the server based implementation distributes fractional components of initial luminance levels to destination gray levels based upon random or average neighborhood gray level techniques (pick one).

Specification of an invalid source IOB, an already in use resource id for the destination IOB, or an invalid IDC generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. Specification of an unconstrained or polychromatic IOB generates a MATCH error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.5.3 Separate Spectral Components

```
XIEP_CHROMINANCE_SEPARATE

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        iob_id3                         XIT_IOB
        iob_id4                         XIT_IOB

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_CHROMINANCE_SEPARATE separates the three planes which make up a three plane multispectral image into three separate monochromatic images. These images represent the red, green, and blue components of the original image.

IOB_id1 is the id of the source image object. IOB_id2, iob_id3, iob_id4 are the client specified ids to be assigned to the newly created image objects.

The input IOB maybe constrained or unconstrained. The output IOB's will be of the same type (constrained or unconstrained) as the input. The input image must be poly chromatic. The output images will be mono chromatic.

Specification of an invalid source IOB, or already in use resource id's for the destination IOB's generates an IDChoice error. Inability of the server to allocate the new IOB's generates an ALLOC error. Specification of a mono chromatic input image generates a MATCH error.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.5.4 Combine Spectral Components

```
XIEP_CHROMINANCE_COMBINE

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        iob_id3                         XIT_IOB
        iob_id4                         XIT_IOB

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_CHROMINANCE_COMBINE combines the three planes which make up a three plane multispectral image into a single multispectral image. The three component images represent the red, green, and blue components components of the image.

IOB_id1, iob_id2, and iob_id3 represent the three color planes which are to be combined to form the composite output multispectral image. IOB_id4 is the client specified id of the output multispectral image object.

All input IOB's must be of the same type (constrained or unconstrained). The output IOB will be of the same type. All input images must be mono chromatic. The output image will be poly chromatic.

Specification of an invalid source IOB, or an already in use resource id for the destination IOB generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. If input IOB's are of different types (constrained and unconstrained), or are not all mono chromatic, a MATCH error occurs.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.5.5 Dither

```
XIEP_DITHER

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB
        levels                          LISTofCARD32

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, VALUE, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_DITHER is used to reduce the number of quantization levels. Dithering accomplishes this by affecting adjacent pixels (area affect) to make up for lack of depth resolution. In essence, it spreads out the value of a pixel into the surrounding area so that the visual affect of multiple levels is preserved.

IOB_id1 is the id of the source image object. IOB_id2 is the client specified id to be assigned to the newly created (resulting) image object. Levels is the number of levels (on a component by component basis) the source image should be dithered down to.

The input IOB must be constrained. The output IOB will be constrained.

Note that because dithering "spreads out" the value of a pixel into the surrounding areas in order to "fool the eye" into perceiving more shades of gray or colors than are actually being used, it generates an image which may be unsuitable for further display processing. While the IOB generated can be operated upon by other XIE functions, the results may prove less than meaningful. As such, dither is often one of the last steps before display.

Specification of an invalid source IOB, or an already in use resource id for the destination IOB generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. If levels is greater than or equal to the current quantized state of the IOB, a VALUE error is generated. If the source IOB is not of type constrained, a MATCH error occurs.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.4.5.6 Luminance

```
XIEP_LUMINANCE

        iob_id1                         XIT_IOB
        iob_id2                         XIT_IOB

        =>

        none

    service class value:                1
    errors:                             IDChoice, ALLOC, MATCH
    events:                             XIE_COMPUTATION_COMPLETE
```

XIEP_LUMINANCE is used to extract or derive the luminance component of a color (polychromatic) image. This function is considerably different than the separate spectral component as the luminance of a color image is derived from the color image by either a color space transform (such as to LUV) or by algorithmic combination of the spectral components (RGB), and not simply by taking one of the spectral components (such as R).

IOB_id1 is the id of the source polychromatic image object. IOB_id2 is the client specified id to be assigned to the newly created (resulting) monochromatic luminance image object.

The input IOB must be both polychromatic and constrained. The output is monochromatic and constrained.

Specification of an invalid source IOB, or an already in use resource id for the destination IOB generates an IDChoice error. Inability of the server to allocate a new IOB generates an ALLOC error. If the source IOB is not polychromatic and of type constrained, a MATCH error occurs.

Client's which request notification are notified of the operation's completion by an XIE_ COMPUTATION_COMPLETE event.

7.5 Statistics Services

Statistical functions allow the client to ask the server to determine certain statistics for an IOB. Statistical functions do not affect the contents of the IOB. rather, they simply report statistical information about the IOB contents back to the client. Statistical information includes component distributions (histogram), moment data, and local area data.

All statistics generation can be done under the control of an IDC. The IDC is used to control the subset of the complete IOB which will contribute to the statistical calculation.

Statistical protocol functions which generate either statistical distributions or local area data, return this information to the client. Statistical protocol functions which calculate moments generate a new (server based) IOB which contains the moment data. This allows the moment data to be queried like any other image, or to be used in server based enhancement processing.

Statistics functions include:

- XIEP_CALC_HISTOGRAM - get the histogram of an image
- XIEP_POINT_STATISTICS - get statistics of an image
- XIEP_AREA_STATISTICS - create a statistical image

7.5.1 Component Distribution

```
XIEP_CALC_HISTOGRAM

        iob_id                          XIT_IOB
        idc_id                          XIT_IDC
        by_component                    BOOL

        =>

        histogram                       LISTofLISTof hist_entry

    service class value:                1
    errors:                             IDChoice, MATCH
    events:                             none

where hist_entry is:

           [
            index        LISTofCARD32
            number       XIT_FLOAT
           ]
```

XIEP_CALC_HISTOGRAM generates a histogram of the component values of the subset of the IOB identified by the IDC. The component distribution reflects the population of all possible component states for a given IOB based upon actual data content. The component distribution can be determined for a bitonal monochromatic image (black and white), continuous tone monochromatic image (gray scale), or continuous tone polychromatic image (color). For polychromatic images, the component distribution can be generated on either a per channel (spectral component) basis (ie. 3 separate distributions) or for the distribution in the three space (ie. 1 combined distribution) with the by_component switch controlling the option.

Note that the size of the list length entry (first field in the list) for the second list in histogram is CARD32 (rather than the default CARD8) to allow for the potentially large number of histogram entries.

IOB_id is the id of the source IOB. IDC_id is the id of the IDC of type ROI or CPP to be used in determining the subset of the image for which the distribution will be generated.

The input IOB must be of type constrained. It may be mono or poly chromatic.

The function returns the histogram to the caller. The histogram is a list of histogram entries where each entry consists of the index or value followed by the number of pixels found with that index or value. For color images where the by_component request dictates three separate distributions, the index is a single value, while for full organization (by_component = false), the index is a list of three values (in essence the color three space coordinates).

Non existent IOB or IDC generates an IDChoice error. Specification of an input IOB of type unconstrained generates a MATCH error.

7.5.2 Point Statistics

```
XIEP_POINT_STATS

    iob_id          XIT_IOB
    x               CARD32
    y               CARD32
    height          CARD8
    width           CARD8

    ->

    Pixel           LISTofXIT_FLOAT
    Minimum         LISTofXIT_FLOAT
    Maximum         LISTofXIT_FLOAT
    Mean            LISTofXIT_FLOAT
    Std_dev         LISTofXIT_FLOAT
    Variance        LISTofXIT_FLOAT

    service class value:          1
    errors:                       IDChoice, VALUE
    events:                       none
```

XIEP_POINT_STATS is designed to allow a client to determine local area statistical information from image data at and surrounding a specified pixel within an IOB. This information is returned by the function to the client.

Iob_id is the id of the source IOB, x and y locate the pixel in question within the IOB. Height and width determines the size of a rectangular area around the pixel to use in the statistical calculations.

The function returns the following information: Value is the value of the pixel at the specified location, minimum is the minimum pixel value in this area, maximum is the maximum pixel value in the area, mean is the mean pixel value for this area, std_dev is the standard deviation of the pixel distribution for this area, and variance is the variance of the pixel distribution for this area.

The input IOB may be either constrained or unconstrained, and mono or poly chromatic. The resulting data is always in float format in order to handle the unconstrained IOB case. Each return item has an entry for each spectral component. Wire transport formats (Band_ Interleaved_by_Pixel, ...) do not apply.

Specification of a non existent IOB generates an IDChoice error. Specification of out of range x and/or y locations generates a VALUE error.

In essence, XIEP_POINT_STATS is a degenerate case of XIEP_AREA_STATS. It differs from XIEP_AREA_STATS in that the function returns the limited statistical information about the area surrounding a single pixel rather than generating a new server IOB containing statistical information about the area surrounding all pixels. Users who only want the pixel value should use the XIEP_GET_PIXEL function.

7.5.3 Area Statistics

```
XIEP_AREA_STATS

    src_iob          XIT_IOB
    dst_iob          XIT_IOB
    idc_id           XIT_IDC {ROI, CPP}
    edge_size        CARD8
    stat_type        XIT_STAT_TYPE

    ->

    none

    service class value:          1
    errors:                       IDChoice, ALLOC, MATCH
    events:                       XIE_COMPUTATION_COMPLETE
```

XIEP_AREA_STATS is designed to allow a client to determine statistical information about a subset of image data within an image, and store this information locally within a new server IOB (ie. create a statistical image).

Src_iob is the id of the source server IOB resource, dst_iob is the id of the destination server IOB resource which will contain the statistical information, idc_id is the id of a server IDC resource of type ROI or CPP, edge_size is the size of the local area square region surrounding each pixel to be used in the statistical operation, stat_type is the type of statistical image to be generated {minimum, maximum, mean, std_dev, variance}.

The function calculates the statistical information on a pixel by pixel basis based upon the local area surrounding pixels and stores the resulting information in the new IOB. The edge_ size parameter determines how large the local area is.

The input IOB maybe either constrained or unconstrained. The output IOB will be of the same type as the input IOB (constrained or unconstrained). The input may only be mono chromatic, and the output will be mono chromatic.

Specification of a non existent source IOB or a resource id for the destination IOB which is already in use generates an IDChoice error. Specification of a non existent IDC or an IDC of type TMP (template) generates an IDChoice error. Inability of the server to allocate the new resource results in an ALLOC error. Specification of a poly chromatic source image results in a MATCH error.

An XIE_COMPUTATION_COMPLETE event occurs when the destination IOB is created and filled with the resulting statistical data.

7.6 Process Control Services

Process Control class services provide the ability to define and apply pipelines containing transport, enhancement, and rendition operations. In addition to the resource functions used to create and query pipelines, Process Control protocol functions include:

- XIEP_ADD_SEGMENT {n} - adds a processing segment to a pipeline
- XIEP_EXECUTE_PIPE {n} - begins execution of a pipeline
- XIEP_KILL_PIPE {n} - terminates (prematurely) execution of a pipeline

7.6.1 Adding Segments

```
XIEP_ADD_SEGMENT

    ipc_id           XIT_IPC
    name             LISTofSTRING8
    function         XIT_XIEFUNCTION
    parm_bindings    LISTofBINDING

    ->

    none

    service class value:          1
    errors:                       IDChoice, VALUE
    events:                       none

where

  BINDING is:

   [
    name                          STRING8
    number                        CARD32
   ]
```

XIEP_ADD_SEGMENT adds a processing segment to the end of a pipeline.

IPC_id the id of the server pipeline resource to which the processing segment will be added. Name is a client assigned name for this segment which will be used by other segments in this pipeline to reference parameters from this segment. Function specifies which of the XIE protocol functions (that are allowed in pipelines) will be used as this pipeline segment. Note that the function choice implicitly determines this pipeline segment's i/o parameter types and numbering.

Parm_bindings describe where this segments inputs will come from and outputs will flow to. There must be as many entries in the list as their are parameters to the XIE protocol function which makes up this segment. Within parm_bindings, name is the name of a prior segment in the pipeline, of the pipeline itself, or the atoms "CONSTANT" or "NULL". For segment inputs, if the input is to come from the pipeline global inputs, the pipeline name should be used and the number is the number of the pipeline global input. If input is to come from an upstream segment, the upstream segment name should be used, and the number is the number of that segments output parameter. If input is a constant, number should be the value of the constant. For segment outputs, if the output is to flow from the pipeline, name should be the name of the pipeline and number the number of one of its output parameters. If the output will be used downstream, or not at all, the name should be set to "NULL".

The following rules apply to pipeline processing segments and their input/output parameters:

1. Segments can only be pipelinable functions (see protocol summary appendix).
2. Segments are added sequentially (no substitution/deletion/modification).
3. Segment input/output parameters can only be bound in the following ways:
   - Input Parameters:
     1. To a constant specified at segment add time.
     2. To a global pipeline input parameter.
     3. To an upstream (previously added) segment output parameter.
   - Output Parameters:
     1. To a global pipeline output parameter.
     2. Left unbound (see below)
4. Intermediate pipe results (iob's etc) not bound to a pipe output are temporary

If the IPC_id is not a valid IPC, or the function is not a pipelinable XIE protocol function, an IDChoice error occurs. If a reference is made to a global pipeline of pipeline segment parameter (index) which does not exist, a VALUE error occurs.

7.6.2 Executing a Pipe

```
XIEP_EXECUTE_PIPE

    ipc_id          XIT_IPC
    input_parms     LISTofparameters
    output_parms    LISTofparameters

    ->

    none

    service class value:          1
    errors:                       IDChoice, MATCH
    events:                       XIE_PIPELINE_COMPLETE
```

XIEP_EXECUTE_PIPE starts the execution of a pipeline.

Input and output parameters are in essence the actual parameters which will be bound to the formal parameters of the pipeline (established at resource creation time with resource type IPC).

If the first segment of the pipeline is an XIEP_PUT_STREAM, the pipeline execution will be stalled until the client executes an XIEP_PUT_STREAM into the designated IOB. As data arrives, it will flow into the pipeline.

If the IPC_id does not identify a valid IPC (one which exists, has at least one processing segment, and is not in state running), an IDChoice error occurs. If any parameter (input or output) is of the wrong type, a MATCH error occurs.

7.6.3 Terminating a Pipe

```
XIEP_KILL_PIPE

    ipc_id          XIT_IPC

    ->

    none

    service class value:          1
    errors:                       IDChoice
    events:                       XIE_PIPELINE_COMPLETE
```

XIEP_KILL_PIPE terminates the execution of a pipeline. IPC_id the id of the server pipeline resource whose execution the client wishes to terminate.

Terminating the execution of a pipeline results in the termination of execution of all pipeline processing elements. All intermediate data and state information is lost as is all pipeline output.

Specification of a non existent source IPC or one which is not currently in the process of execution results in an IDChoice error.

An XIE_PIPELINE_COMPLETE event occurs when the pipeline is terminated.

7.7 Display Services

As the core X protocol provides substantial machinery to facilitate pixmaps and the X image extension works to produce (through rendition) pixmaps, the only display services in XIE deal with time sequences, something not provided for under core X.

```
XIEP_DISPLAY_SEQUENCE

        window_id       WINDOW
        gc_id           GCONTEXT
        frame_count     CARD16
        frame_timing    CARD16
        frame_list      LISTofPIXMAP
        forward         BOOL

        =>

        none

    service class value:        3
    errors:                     IDChoice, VALUE
    events:                     XIE_SEQUENCE_COMPLETE
```

XIEP_DISPLAY_SEQUENCE is used to display a sequence of prerendered image frames which are stored as server pixmaps. The client has control over the order of frames, time delay between frame display, and direction of frame sequence play.

Window_id is the window in which to display the sequence. GC_id is the id of the GC used in the display operation. Frame_count is the number of frames in the sequence. Frame_timing controls the delay time in milliseconds between sequential frame displays. Frame_list is the list of PIXMAP id's that constitute the sequence of frames. Forward is a boolean which controls the direction of play of the sequence. Note that the size of the list length entry (first field in the list) for frame_list is CARD16 (rather than the default CARD8) to allow for a long sequence of pixmaps.

Specification of an invalid window_id or an invalid pixmap_id generates an IDChoice error. Specification of frame_timing which is less than that which the server can provide generates a VALUE error.

For clients which have requested notification of end of sequence events, an XIE_SEQUENCE_ COMPLETE event occurs when the last frame of a sequence is displayed.

7.8 Event Services

X Image Extension Protocol event services provide for the selection and query of image extension specific events.

7.8.1 Select Events

```
XIEP_SELECT_EVENTS

        event_mask      CARD16

        =>

        none

    service class value:        0
    errors:                     VALUE
    events:                     none
```

XIEP_SELECT_EVENTS is used by the client to select which events it wishes to be notified of. Event_mask is a bit map mask where each bit corresponds to the value of an event code.

Specification of an invalid event_mask generates a VALUE error.

7.8.2 Query Events

```
XIEP_QUERY_EVENTS

        event_mask      CARD16

        =>

        event_mask      CARD16

    service class value:        0
    errors:                     none
    events:                     none
```

XIEP_QUERY_EVENTS is used by the client to determine which events it has expressed interest in being notified of.

Event_mask on the input side communicates which events the client is interested in knowing its notification status, while on the destination side, it communicates the server's record of which events the client will be notified of. In both cases, it is a bit map mask where each bit corresponds to the value of an event code.

Chapter 8

EVENTS

All events return a message with a common prefix as described in the Event Protocol Message Format section. Included in this information is the event type code, and the request sequence number.

The following sections define the details of each of the image extension events. All X Image Extension event names begin with the prefix XIE_. Event codes are defined in the protocol code summary appendix.

8.1 COMPUTATION COMPLETE

```
XIE_COMPUTATION_COMPLETE

      src_iob           XIT_IOB
      dst_iob           XIT_IOB
```

An XIE_COMPUTATION_COMPLETE event occurs when client requested process protocol functions completes (see protocol specifications). Specifically, it occurs when all the image data that makes up the complete contents of the destination IOB which results from the computation has been established by the server.

The event information includes the the id's of both source and destination IOB's.

Only clients which have requested notification of this event are notified.

8.2 PIPELINE COMPLETE

```
XIE_PIPELINE_COMPLETE

      ipc_id            XIT_IPC
```

An XIE_PIPELINE_COMPLETE event occurs when client requested process pipeline completes (see protocol specifications). Specifically, it occurs when all the image data that makes up the input image(s) has been processed by the last stage in the pipeline.

The event information includes the the id of the pipeline (IPC id).

Only clients which have requested notification of this event are notified.

8.3 SEQUENCE COMPLETE

XIE_SEQUENCE_COMPLETE

An XIE_SEQUENCE_COMPLETE event is generated when a server completes displaying a time sequence of image frames.

Chapter 9

APPENDICES

9.1 XIE Protocol Summary

The following is a summary of the XIE protocol functions categorized into their various service classes. A {p} following the function indicates that it can be used as a pipeline element, while a {n} indicates that it can not be used as a pipeline element. *Session Services*

- XIEP_INIT_SESSION {n} - begins a client-extension server session
- XIEP_SET_DEFAULTS {n} - establish/modify operational defaults
- XIEP_QUERY_DEFAULTS {n} - query operational defaults
- XIEP_TERM_SESSION {n} - ends a client-extension server session

Resource Services

- XIEP_CREATE_RESOURCE {n} - create a new server resource object
- XIEP_COPY_RESOURCE {n} - copy contents from source to destination resource
- XIEP_DELETE_RESOURCE {n} - delete an existing server resource object
- XIEP_QUERY_RESOURCE {n} - query an existing server resource object
- XIEP_IMPORT_IMAGE {n} - create a new server IOB resource object
- XIEP_EXPORT_IMAGE {p} - create a new server PIXMAP resource object

Transport Services

- XIEP_SET_TRANSPORT {n} - establishes the required transport context information
- XIEP_GET_TILE {p} - transports random tiles of uncompressed image data from server to client
- XIEP_PUT_TILE {p} - transports random tiles of uncompressed image data from client to server
- XIEP_GET_STREAM {p} - transports a stream of compressed or uncompressed image data from server to client
- XIEP_PUT_STREAM {p} - transports a stream of compressed or uncompressed image data from client to server

Process Services

- XIEP_POINT {p} - applies a point (lut) transform to an image
- XIEP_AREA {p} - applies are operators to an image
- XIEP_ARITH {p} - performs arithmetic between two images
- XIEP_COMPARE {p} - compare two images
- XIEP_LOGICAL {p} - operates on the individual bit planes of an image
- XIEP_MATHF {p} - applies mathematical functions to an image

- XIEP_SCALE {p} - scale image in x and y directions
- XIEP_ROTATE {p} - rotate image around z axis
- XIEP_MIRROR {p} - rotate image around x and/or y axes
- XIEP_CROP {p} - extract a subset of an image
- XIEP_TRANSLATE {p} - shift and combine two images

- XIEP_CONSTRAIN {p} - apply a quantization model
- XIEP_MATCH_HISTOGRAM {p} - transform the image to have the specified histogram form
- XIEP_CHROMINANCE_SEPARATE {p} - separate a poly chromatic image into multiple monochromatic images
- XIEP_CHROMINANCE_COMBINE {p} - combine multiple mono chromatic images into a poly chromatic image
- XIEP_DITHER {p} - reduce number of quantization levels through an area technique
- XIEP_LUMINANCE {p} - extract the luminance component from a poly chromatic image

Statistics Services

- XIEP_CALC_HISTOGRAM {n} - get the histogram of an image
- XIEP_POINT_STATISTICS {n} - get statistics of an image
- XIEP_AREA_STATISTICS {n} - create a statistical image

Process Control Services

- XIEP_ADD_SEGMENT {n} - adds a processing segment to a pipeline
- XIEP_EXECUTE_PIPE {n} - begins execution of a pipeline
- XIEP_KILL_PIPE {n} - terminates (prematurely) execution of a pipeline

Display Services

- XIEP_DISPLAY_SEQUENCE {n} - display a time sequence of pixmaps

Event Services

- XIEP_SELECT_EVENTS {n} - select which events the client wishes to be notified of
- XIEP_QUERY_EVENTS {n} - query which events the client will be notified of

9.2 Recommended Server Resource Contents

This section describes the client known contents of the various image extension server resources. These are the resource object fields which the client can reference in various resource protocol operations. Some fields can be initialized, queried, and modified by the client through object creation, modify, and query functions, while others can only be initialized and queried, and still others can only be queried. This last class of fields are initialized and maintained solely by the server, and are in essence read-only for the client

A table within each subsection defines the field identifier, allowable forms of access, and value binding. Particulars regarding image data object fields can be found in the appendix on image data attributes and organization. Details of other resources and their contents can be found in the resource creation protocol functions and other protocol functions which use the resources.

9.2.1 IOB

The atomic unit of revisable form image data residing on the server and addressable by the client is the image object or IOB. An IOB represents a single revisable form image (one which can be modified and prepared for device specific display). An IOB is intended to contain a single two-dimensional still-frame image of bitonal, continuous tone, or multispectral data. IOB's are not intended for composite image collections or temporal image sequences (multiple 2d frames associated with spatial or temporal layout mechanisms). Multiple IOB's should be used by applications to build composite static image displays and time sequences of images under application control. IOB's are self contained: no nesting or external references are allowed.

Each image object (IOB) contains image attributes and the image data itself. The attributes consist of information necessary to interpret the image data and useful in guiding default rendition and presentation. The attributes are initialized by the client at image object creation time. The image data can only be initialized through image data wire transport functions. All attributes can be directly queried by the client, but the image data can not be directly queried. It can only be read via image data wire transport functions.

IOB attributes fall into two classes: those that are server private, and those that are public and queryable by the client. The private attributes can not be accessed by the client in anyway and are not described in this document. The public attributes are initialized by the client and server at image object creation time. Once the IOB has been created, these attributes are "read only" for the client. The only modification of these attributes derives from client directed server processing of the image.

Image primitive and presentation attributes are stored within the IOB and are initialized and queryable by the client. Format (organization and encoding) information is not stored in the IOB (refer to appendix on image data attributes and organization). Format information is simply used during wire transport.

Field id	Field code	Field type
IOB$PIXELS_PER_SCANLINE	0	CARD32
IOB$NUMBER_OF_SCANLINES	1	CARD32
IOB$NUMBER_OF_SPECTRAL_COMPONENTS	2	CARD8
IOB$SPECTRAL_COMPONENT_MAPPING	4	XIT_SPECTRAL_COMPONENT_MAPPING
IOB$PIXEL_PROGRESSION	5	XIT_PIXEL_PROGRESSION
IOB$LINE_PROGRESSION	6	XIT_LINE_PROGRESSION
IOB$ASPECT_RATIO	7	XIT_FLOAT
IOB$POLARITY	8	XIT_POLARITY
IOB$CONSTRAINED	9	BOOL (query only)
IOB$CONSTRAIN_MODEL	10	CARD8 (query only)
IOB$CONSTRAIN_BITS_PER_COMPONENT	11	LISTOFCARD8 (query only)

The image data field of the IOB is only accessible to the client through image data wire transport protocol function, and not directly through resource object functions.

Other than the implications of constrained/unconstrained and mono/polychromatic, the organization of the image data in the IOB is server private. The server may choose to represent the revisable form image data within the IOB in anyway it desires including the same way in which the core server represents PIXMAPS. The only constraint is that upon request from the client, the server must be able to reproduce ALL image data which comes across the wire in the first place (ie. if the client creates an IOB, fills it with image data, and requests the server to send the data back in the same way, the data which comes back must be identical to what went across in the first).

9.2.2 IDC

Control over the selection of the subset of a revisable form image object which will be processed by the server, as well as the specification of operand data for area operations is accomplished through the use of image domain control contexts (IDCs).

The subsetting process serves to identify either fully connected subregions of an image which will be processed, or individual pixels and disconnected subregions.

Area operations involve specification of both area operands to be used in conjunction with the area image data surrounding the current pixel, and the spatial relationship or registration of those area operands (both spatial registration and operand shape).

A region of interest defines a rectangular subregion of an image object (less than or equal to the image object in size). The specification is accomplished by locating the upper left and lower right hand corners of the rectangle relative to an image object. ROI field values are initialized at resource creation time. They can be queried or modified at any time by the client. They are never modified by the server.

Field id	Field Code	Field type
IDC$ROI_UL_X	0	CARD16
IDC$ROI_UL_Y	1	CARD16
IDC$ROI_LR_X	2	CARD16
IDC$ROI_LR_Y	3	CARD16

A control processing plane defines the points within a rectangular subregion of an image object (less than or equal to the image object in size) that are subject to processing. The specification is accomplished via the use of a 1 bit deep PIXMAP which is positioned against the image object by locating the upper left hand corner relative to an image object. A PIXMAP value of 1 implies that the image data at that point is subject to processing while a value of 0 implies that it is not. The results of the image operator at these points (selected and not) are determined by the operator on a case by case basis. CPP field values are initialized at resource creation time. They can be queried or modified at any time by the client. They are never modified by the server.

Field id	Field Code	Field type
IDC$CPP_UL_X	4	CARD16
IDC$CPP_UL_Y	5	CARD16
IDC$CPP_DATA	6	PIXMAP

A template defines the points within a rectangular subregion which will be selected for use in an area operation, and the values of an operand to be used in the operation for each of these points. The template does not define its spatial relation to the image object. Rather, it specifies the alignment of its selection and operand "mask" to what ever the current pixel is. This is done by specifying the location of the current pixel in the two dimensional selection / operand grid. This allows for asymmetric and sparsely populated template grids. TMP field values are initialized at resource creation time. They can be queried or modified at any time by the client. They are never modified by the server.

Field id	Field Code	Field type
IDC$TMP_ENTRIES	7	CARD8
IDC$TMP_CENTERX	8	CARD8
IDC$TMP_CENTERY	9	CARD8
IDC$TMP_DATA	10	LISTofXIT_FLOAT

9.2.3 IPC

TBS

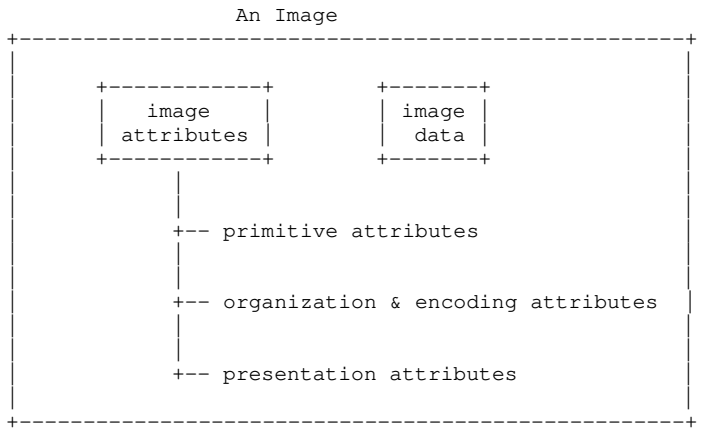
9.3 Image Data Attributes & Organization

In the X Image Extension, the client may keep image data within itself in any fashion it chooses. The client may keep any set of attributes, partitioned or not, and any image data organization. It may also choose to exchange that data and those attributes with another application in any fashion it desires. Those issues are outside of the scope of the X Image Extension.

What the X Image Extension does focus on (specify) is the transport of image data and attributes across the X wire. The ground rules are simple: the protocol specifies what attribute information must be exchanged and when, and the multiple allowable data transport formats. The client gets to choose from these various formats and the server complies at data transport time. The client does not get to choose how the server represents or organizes the image data (just as the server does not control how the client does it). The server may be lazy and simply maintain the organization as it came across the wire, or may restructure the data to meet its needs. While the requirement (see protocol section) that the server be able to turn around and give image data back precisely as it received it (without loss) may be a powerful incentive to maintain the wire transport data organization, it is not a requirement that the server maintain the data internally in that format.

This appendix addresses image data attributes, their logical partitioning, and image data transport formats. The various attributes are partitioned and described along with the transport formats. The parameter types for these attributes are described either in the parameter types chapter (for complex types), or in-line along with their usage in the protocol functions chapter (transport services section).

Image attributes are partitioned into three components: primitive attributes, organization and encoding attributes, and presentation attributes. Primitive attributes relate to a PCM reference model for the image. Organization and encoding attributes are essential for interpreting the image data at transport time and go to the heart of the issue of transport formats. Presentation attributes relate to the 'intended' or original presentation established at capture or synthesis time.



9.3.1 Primitive Attributes

Primitive image attributes include information about a PCM reference model version or state of the image (which may not exist during transport). Primitive image attributes are communicated by the client to the server at IOB creation time, are queryable by the client at any time, and modifiable only by the server (not the client).

Primitive attributes include:

- Pixels-Per-Scanline

  The total number of pixels per scan line. This value is not necessarily the total number of bits per scan line.

- Number-of-Scanlines

  The total number of scan lines in the image.

- Number-of-Spectral-Components

  Specifies the number of spectral bands in an image. Note that this is NOT necessarily the number of logical data planes in the data organization. For an RGB image, this would be 3. For a mono chromatic image, this would be 1.

- Spectral-Component-Mapping

  Designates the correlation between the physical image data and the spectral components of an image. Images may have anywhere between one spectral component (monochromatic) and several spectral components (polychromatic). Monochromatic bitonal (black & white), monochromatic-continuous tone (grayscale), and 3 band polychromatic (color) will be the only supported classes.
  Recognized spectral component mappings include:
  - monochrome
  - color 3 spaces
    - RGB
    - *futures*

9.3.2 Organization & Encoding Attributes

Organization and encoding attributes define the detailed structure of the image data representation during wire transport. Image data will either begin or end up in a client's buffer in this format depending upon the transport direction. Organization and encoding attributes are only relevant in wire transport and are not directly associated with a server based IOB.

9.3.2.1 Encoding Attributes

Encoding attributes describe the image data compression being used during wire transport, and its relationship to the data organization scheme in use. Encoding attributes include:

- Compression-Scheme

  The compression scheme used to encode the image data during transport. Only standardized or well recognized and accepted compression schemes are supported. These include:

  - PCM (raw bitmap - no compression)
  - CCITT Group 4 two dimensional
  - ISO Discrete Cosine Transform

There are ground rules for the choice of compression scheme and its application based upon data organization.

The fax compression scheme (G42D) can only be applied to bitonal images (any value of Component-Space-Organization as long as the total bits per pixel = 1), or to continuous tone images (gray scale and color) on a bit-plane by bit-plane basis (Component-Space-Organization must be bit_interleaved_by_plane).

Discrete Cosine Transform compression can be applied to bitonal, continuous tone gray scale, and continuous tone color images with Component-Space-Organization equal to band_ interleaved_by_plane or bit_interleaved_by_plane. In the former case, the compression scheme is applied on a spectral component basis, while in the latter case, it is applied on a bit plane by bit plane basis.

9.3.2.2 Component Space Attributes

Image data may represent information deriving from a number of chrominance bands (mono through poly chromatic) and luminance quantization levels per band (bitonal through continuous tone). Given any one combination (such as 3 color bands as in R-G-B and 8 bits per band per pixel), the representation of the information can be organized for transport in several ways. This section describes the possible formats and the associated attributes along with low level data formatting information. Component space attributes include:

- Component-Space-Organization

  Designates how the component space data is physically organized. It is defined in terms of pixel component bit storage within and across image logical data planes. In other words, it indicates how image data is organized pixel-by-pixel and spectral-component by spectral-component into logical planes of data. Recognized component-space-organizations include:

  - Band_Interleaved_by_Pixel

    All pixel data for ALL spectral components is collected into a single logical data plane. For example, given an RGB image, the red,

green, and blue bits for one pixel are butted together. The red, green, and blue bits for the next pixel follow.

- Band_Interleaved_by_Plane

  All pixel data for EACH spectral component occupies a separate logical data plane. Thus, for our RGB image, all red bits for all pixels are gathered into a single data plane, followed by all green bits for all pixels, followed by all blue bits for all pixels.

- Bit_Interleaved_by_Plane

  All the pixel bits for each spectral component occupy separate logical data planes. Thus, if our RGB image had 8 bits per spectral component, there would be 24 separate data planes.

Bitonal images (monochromatic) can be organized in any of the three ways (all are identical). Grayscale images (monochromatic) can be organized in two ways: (Band_Interleaved_by_Pixel and Band_Interleaved_by_Plane are identical) or (Bit_Interleaved_by_Plane). Color images (polychromatic) can be organized into any of the three ways.

- Bits-Per-Component

  The number of bits used for each image component by logical data plane independent of component space organization. The sum of Bits-Per-Component for all components must by definition equal the total bits per pixel.

- Pixel-Stride

  The difference in bit addresses between the starting bits of successive pixels by logical data plane. Pixel-stride will typically be equal to the number of bits per pixel stored in a particular plane unless pixel boundary alignment requires fill (pad). Pixel-Stride is meaningful for uncompressed data in any of the 3 organizational formats.

- Scanline-Stride

  The difference in bit addresses between the starting bits of successive scanlines by logical data plane. Scan-Line stride will typically be equal to the number of bits (not pixels) per scanline unless scanline boundary alignment requires fill (pad). Scanline-Stride is meaningful for uncompressed data in any of the 3 organizational formats.

9.3.3 Presentation Attributes

Presentation attributes represent qualitative information that corresponds to visual characteristics of the image data when presented (viewed). Presentation attributes affect how image data is rendered and presented for viewing. They often derive initially from the original object or scene, and the capture or synthesis process. As the capture process involves a digitization and quantization process, the attributes of the capture device/process are starting points for many presentation attributes. Maintaining this information and using it at presentation time ensures fidelity between the original object or scene and the rendered image (ie. the image will look like the original object or scene). An example of ignoring or modifying this information might result in an image displayed upside down. Presentation attributes can be found in the IOB.

A major component of presentation deals with the layout of an image on the screen. For this aspect of presentation, an output device (screen) relative coordinate reference system is necessary. For this purpose, we use the ISO and CCITT standard image orientation. In this system a two dimensional cartesian coordinate system is overlaid on top of the screen surface (the z axis would be normal to the plane and out of the screen). The x axis is placed horizontally aimed towards the users right (user facing screen) and the y axis is aimed vertically. All angles are measured relative to the positive x axis in a counter clockwise direction.

Presentation attributes include:

- Pixel-Progression

  The direction of the pixel path relative to ISO and CCITT standard image orientation. More aptly, the angle between a scan line and the x axis of the standard image orientation coordinate system. Standard values of 0, 90, 180, and 270 degrees should be used allowing for left-to-right, right-to-left, top-to-bottom, and bottom-to-top pixel progressions.

- Line-Progression

  The direction of scan line capture / display across the image plane. More aptly, the direction normal to that of a scan line in which scan lines were captured or will appear. Standard values of 90 and 270 degrees should be used.

- Aspect-Ratio

  The ratio of the distance between pixel centers along the pixel-progression direction and the line-progression direction.

- Polarity

  The manner in which the pixel values represent minimum and maximum intensity.

9.4 Concept and Version Changes

There have been several significant conceptual changes between the draft of the concepts document and this draft (v2.0) of the protocol specification. In addition, there have been substantial changes in content between v1.0 and v2.0 of the protocol specification. This appendix summarizes the conceptual and protocol changes.

9.4.1 Conceptual Changes

Perhaps the most significant conceptual change has to do with the location of the wire relative to the rendition services that the x image extension calls for. In the concept document, high level rendition services were called for on the server side of the wire. The current position is to provide a minimal set of tools on the server side which can be used in rendition processing (and might be facilitated by display accelerators), and leave the direction of rendition models to a client side imaging toolkit. To some extent, this is in support of a general effort we've made to reduce the size of the extension protocol.

A second shift is based upon the recognition that resource types are themselves scarce things. A such, the protocol has moved away from the use of many different types of extension resources towards a minimal set with more in line parameters. In particular, ITC's (image transport contexts), IEC's (image enhancement contexts), and IDC's (image display contexts) are no longer server resources. Instead, the information they contained is exchanged as protocol parameters.

The authors have always felt that logically, image capture devices such as frame grabbers (video digitizers) belong on the client side of the wire, but that their close proximity to display subsystems made it necessary to provide for them on the server side of the wire (through the protocol). While prior drafts have included protocol functions for control of video digitizers, they have been dropped from this draft. The reason for this is that the requirements of the video community are beyond the scope of this effort and it was felt that they could best be served by a separate video extension.

The authors originally intended to provide access to high level image enhancement functions on the server via the protocol. Because of the enormous set of such functions (unbounded), and because architecturally it was felt that high level image function packaging belonged closer to the application, the protocol has shifted to support lower level image functions. The higher level packaging of these functions has been pushed back across the wire to the tool kit level. The X Image Extension Tool Kit document provides an exemplary set of application level image display and enhancement processing services which can be built upon the core X11 and X Image Extension protocols.

The position on server storage devices has not changed - the extension does not acknowledge the existence or usage of such devices (including the issue of storage formats). The use of such devices is at the discretion of server implementors. It is possible that storage devices and formats may appear at the toolkit level.

In an effort to reduce the image extension display services, representation and display of tiled images were removed from the protocol. It was felt that there is sufficient machinery in the core protocol to support this from the client side. Tiled image support is likely to be another issue for the tool kit.

9.4.2 V1.0 - V2.0 Protocol Changes

Many significant protocol changes were made between v1.0 and v2.0. These were based upon the above conceptual changes, comments from reviewers, and a general effort to trim down the protocol extension and make it more compliant with the core x philosophy.

Minimality and Organization of Protocol

We received many complaints regarding the size of the protocol (number of functions), the organization of the protocol (function classes), and the level of complexity of operations required of the server. We undertook to reduce in level and extent the contents of the protocol, as well as to meaningfully reorganize it. The salient points and reasoning are discussed below.

The video functions (those which involved a digitizer) have been completely removed from the specification. Our feeling is that the needs of the video community are better served by the video extension and that a connection between video and digital image processing can occur through pixmaps in the core server.

We have reduced the display functions in the belief that the core display functions are sufficient in many ways and in that more complex display mechanisms can be built upon them and managed from the client side (toolkit). The major exception to this is time sequences of images which we feel require server support to work (time sequences remain).

High level class conversion (rendition) functions were analyzed and a few primitive functions missing from the original draft were identified (most notably dither) and added. The high level functions were then deleted (moved to the toolkit) where they can be built upon the server primitives.

Enhancement and rendition functions were reorganized into process class functions due to their similarity and their generic applicability. Thanks goes to members of SUN who contributed insight into how to meaningfully reorganize these functions.

Across the entire protocol, multiple similar functions which differed only in terms of an operator were collapsed (combined) by making operator selection an operand. No functionality was lost, but the protocol is easier to follow and it may help to organize the implementations.

Use of Core Resources

Unessential extension resources (IEC, IRC, IDC) have been deleted completely in favor of in line parameters.

We've made use of the core resource GC as is where necessary versus reinventing some of its contents in extension specific resources.

Error Codes

We have removed all extension specific error codes and are now making use of core error codes only.

Connect Information

All core X redundant connect time information has been removed and all unnecessary protocol information (such as session_id) has been removed.

We've changed XIT_SUPPORT_VECTOR to XIT_FUNCTION_VECTOR to reflect what functions (as opposed to service classes) are provided by the server. The service class will provide guidelines to the the classification of requests.

IOB Data Formats

It became clear that while positive integral data values for image data on the wire was acceptable, it would be insufficient for many server protocol functions. As such, the extension has defined a "constrained" and "unconstrained" IOB type (which translates roughly to positive integral and positive/negative float - see chapter on server data model). It now includes facilities for managing these data types.

Color Space Support

The initial spec called for multiple color space support on the server (rgb, yuv, cie luv ...). As current display devices are primarily rgb (yuv may begin to appear due to digital tv, but not today), the decision was made to push alternate color space support back across the wire into the toolkit (along with color correction). For now, wire transport and server processing will only support rgb color images.

Pipelines

The pipeline TBS has been replaced with actual protocol functions.

9.5 References

1. X Window System Protocol, Version 11
   Alpha Test, February 1987 - Robert W. Scheifler

2. CCITT T.4

   "Standardization of Group 3 Facsimile Apparatus for Document Transmission"

   Defines general device capabilities, and defines tables
   and algorithms for one- and two-dimensional coding schemes,
   as well as scheme for "uncompressed data."

3. CCITT T.6

   "Facsimile Coding Schemes and Coding Control Functions
   for Group 4 Facsimile Apparatus"

   Defines coding tables and algorithm for two dimension
   encoding used in G4 fax.

4. CCITT T.73

   "Document Interchange Protocol for the Telematic Services"

   Defines document interchange protocol for use above
   the session services layer used by teletex and G4 fax.

5. ISO DPX3

   "Photographic Image Coding"

   Defines algorithms for encoding continuous tone photographic
   images for character protocol terminal services (videotex).

6. Digital Image Processing
   William K. Pratt
   John Wiley & Sons - 1978

7. Digital Image Processing
   Rafael Gonzales and Paul Wintz
   Addison Wesley - 1987

8. Digital Picture Processing
   Azriel Rosenfeld and Avinash Kak
   Academic Press - 1982

9. Image Algebra Project Publications
   Computer Vision Laboratory
   University of Florida, Gainesville

   Department of The Air Force
   Air Force Armament Laboratory
   Eglin Air Force Base, Florida