A

Techniques

Standard and Private Techniques

For some XIE operations, several recognized algorithms or *techniques* offer varying tradeoffs between quality of results and performance. Also, in some cases, different techniques are required due to an image's class or content. In general, techniques that are specified for import and export elements identify image compression schemes, and techniques that are specified for process elements identify various image processing or enhancement algorithms.

Several *standard* techniques are defined in this appendix. Standard techniques are those that are well known to the image processing community. They are further categorized in this document as *required*, *optional*, or *excluded* (see Table B-2). While required techniques must be implemented in all servers, it is expected that most optional techniques will also be widely available. Some techniques are excluded from certain subsets.

Vendors may choose to extend XIE with their own *private* techniques to provide for their particular needs. Therefore, private techniques are likely to be available only on a particular vendor's platforms.

Technique numbers

Standard and private techniques can be differentiated by their technique numbers. The most significant bit of the technique number is zero (0) for standard techniques and one (1) for private techniques. Standard technique numbers are defined in this document (see Appendix C). Private technique numbers are assigned dynamically (the method for generating private technique numbers is chosen by the server implementor). Because private technique numbers are not statically defined, the number to name-string mapping must be obtained on a per-server basis via the **QueryTechniques** protocol request.

Technique names

The name string for standard techniques contains only the name of the algorithm or compression scheme (for example, "ERROR-DIFFUSION" or "CCITT-G32D"). Private technique name strings also include the name of the vendor that owns the rights to the technique (for example, "_PHOTOCO_SQUASH-BITS"). Because the organization name is encompassed by "_" (underscore) characters, the leading underscore provides another means by which standard techniques can be can differentiated from private techniques.

Default Techniques

In some cases, it is appropriate to assign a *default* technique. A default technique is a synonym that must be bound to a standard or private technique. Where there is no required standard technique, a default technique must be defined that is bound to an optional or a private technique. No default is defined in cases where it would be inappropriate (for example, decode techniques describing client data).

Technique parameters

For each occurrence of a technique within an element definition, allowance is made to pass parameters that are specific to the technique. Some techniques have defined parameters, and others have none. For techniques that do have defined parameters, the **QueryTechniques** request can be invoked to determine if the parameters must be supplied. If the parameters are optional and they are omitted by the client, the server will supply implementation specific default values in their place.

If the default technique is requested, no additional parameters are accepted (implementation specific default values will be provided for parameters defined for the technique to which ***Default*** is bound).

Technique information

The **QueryTechniques** request returns information about ***All*** techniques, all ***Default*** techniques, or a group of techniques that are functionally similar (for example, ***Dither***, ***Encode***, ***Geometry***, and so on). The following information is returned about the selected techniques:

needs-parameters	Indicates that the technique requires additional parameters; if false, the technique takes no parameters, or its parameters are optional. If the parameters are optional, they can be totally omitted, otherwise they must all be supplied.
group	Identifies which group the technique belongs to.
number	Specifies the numeric identifier assigned to the technique (MS bit is zero for *standard* techniques, or one for *private* techniques).
speed	Specifies the server's assessment of the speed of this technique relative to other techniques in the same group (where 0 is the slowest, and 255 is the fastest).
name	Identifies the technique name string in the form: <STANDARD-TECHNIQUE-NAME> or _<ORGANIZATION-NAME>_<PRIVATE-TECHNIQUE-NAME>

The remainder of this appendix provides a complete description of each technique and its parameters.

Color Allocation Techniques

Color allocation techniques allocate or match colors or gray shades from a COLORMAP. Cells allocated from a dynamic COLORMAP are recorded in a **ColorList**. If the COLORMAP is static, image pixels can only be matched to existing COLORMAP entries; no allocations are made and no **ColorList** needs to be specified.

AllocAll

Parameters

fill: CARD32

AllocAll allocates a read-only COLORMAP cell for each new pixel found. If the COLORMAP runs out of cells, the remaining new pixels are mapped to *fill*. A **ColorAlloc** event can be sent if it is necessary to use *fill*. ***AllocAll*** is only appropriate for dynamic COLORMAPs and requires that the number of discrete image pixels *fit* within the size of the COLORMAP to avoid running out of cells.

Match

Parameters

match-limit: XieTypFloat
gray-limit: XieTypFloat

Match allows a trade-off between image fidelity and COLORMAP usage via a pair of granularity parameters [[1]]. The highest priority is given to allocating read-only cells in a sequence that provides an even distribution of pixels throughout the colorspace. Secondary priority is given to the frequency of usage of image pixels. Any image pixel that is a close enough match to an existing read-only cell will share that cell (where "close" is determined by the granularity controls). For other image pixels, new read-only allocations are made. When no more cells are available, each remaining image pixel is matched to the closest read-only cell. ***Match*** is appropriate for both static and dynamic COLORMAPs. For the sake of computational efficiency, the number of discrete image pixels should not exceed the size of the COLORMAP.

Match-limit and *gray-match* control the allocation of colors and gray shades, respectively. The minimum value (0.0) specifies exact matches (within the limits of the COLORMAP). The maximum value (1.0) encompasses the entire colorspace within which no new cells are allocated. A **ColorAlloc** event can be sent if the COLORMAP runs out of cells.

Requantize

Parameters

max-cells: CARD32

Requantize first reduces the total number of discrete pixels values in the image to be no more than a specified number and, then, allocates the resulting pixels values as read-only cells from the COLORMAP. One method of accomplishing this reduction process can be found in [[2]].

Max-cells specifies the maximum number of COLORMAP allocations to allow. If *max-cells* is zero or greater than the number of unallocated COLORMAP cells, the reduction algorithm will restrict its output to the number of free cells. A **ColorAlloc** event can be sent if the number of pixels had to be restricted to a lesser number than *max-cells* due to a lack of free COLORMAP cells. ***Requantize*** is only appropriate for dynamic COLORMAPs.

---

[1] A full description of the color matching algorithm is available upon request from author Shelley.

[2] "Color image quantization for frame buffer display", by P. SHeckbert, *Computer Graphics* (ACM SIGGRAPH '82Conf. Proc.), vol. 16, no. 3, page 297.

Constrain Techniques

ClipScale

Parameters

input-low: XieTypConstant
input-high: XieTypConstant
output-low: XieTypLevels
output-high: XieTypLevels

For each band, output pixels will be clipped to the range [*output-low, output-high*].

If *input-low* is less than *input-high*, then all pixels less than or equal to *input-low* will map to *output-low,* and all pixels that are greater than or equal to *input-high* will map to *output-high.* All intermediate pixel values are scaled proportionately to the output range. Nonintegral output values are rounded to the nearest integer.

If *input-low* is greater than *input-high* then all pixels that are greater than or equal to *input-low* will map to *output-low*, and all pixels that are less than or equal to *input-high* will map to *output-high*. All intermediate pixel values will be linearly mapped to the output range such that *input-low* maps to *output-low,* and *input-high* maps to *output-high.* Nonintegral output values are rounded to the nearest integer.

If *input-low* equals *input-high*, or if *output-low* or *output-high* exceeds *levels*-1, a ***FloTechnique*** error is generated.

HardClip

Parameters

< none >

HardClip constrains the data such that all negative pixels are set to zero, and all pixels greater than *levels*-1 are set to *levels*-1. All other pixels are rounded to the nearest integer value.

Convert From RGB

CIELab

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params: <technique-dependent>

Matrix is a 3x3 RGB to ***CIEXYZ*** conversion matrix (the source white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the output data. *White-params* is the list of parameters required by *white-adjust*.

The input **DataType** can be ***Constrained*** or ***Unconstrained***; the output **DataType** is always ***Unconstrained***. When the input is ***Constrained***, the data are normalized to the range [0, 1] (that is, scaled by 1/(*levels* **-** 1) prior to the conversion.

CIEXYZ

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params:<technique-dependent>

Matrix is a 3x3 RGB to ***CIEXYZ*** conversion matrix (the source white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the output data. *White-params* is the list of parameters required by *white-adjust*.

The input **DataType** can be ***Constrained*** or ***Unconstrained***; the output **DataType** is always ***Unconstrained***. When the input is ***Constrained***, the data are normalized to the range [0, 1] (that is, scaled by 1/(*levels* **-** 1) prior to the conversion.

YCbCr

Parameters

luma: XieTypConstant
levels: XieTypLevels
bias: XieTypConstant

Luma represents the proportions of red, green, and blue in the luminance band, ***Y***. *Levels* determines the output *levels* if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Bias* is used to add an offset to the output pixels values.

Source data may be ***Constrained*** or ***Unconstrained***; the output *type* will match.

YCC

Parameters

luma: XieTypConstant
levels: XieTypLevels
scale: XieTypFloat

Luma represents the proportions of red, green, and blue in the luminance band, ***Y***. *Levels* determines the output *levels* if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Scale* is used to compress the output pixels (a typical value is about 1.35 to 1.4 in the literature).

Convert To RGB

CIELab

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params: <technique-dependent>
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Matrix is a 3x3 ***CIEXYZ*** to RGB conversion matrix (the target white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the source data prior to conversion. *White-params* is the list of parameters required by *white-adjust*. *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

The input **DataType** must be ***Unconstrained***; the output **DataType** is also ***Unconstrained***.

CIEXYZ

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params: <technique-dependent>
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Matrix is a 3x3 ***CIEXYZ*** to RGB conversion matrix (the target white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the source data prior to conversion. *White-params* is the list of parameters required by *white-adjust*. *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

The input **DataType** must be ***Unconstrained***; the output **DataType** is also ***Unconstrained***.

YCbCr

Parameters

luma: XieTypConstant
levels: XieTypLevels
bias: XieTypConstant
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Luma represents the proportions of red, green, and blue in the luminance band, ***Y***. *Levels* determines the output *levels* if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Bias* is used to remove any offset from the source pixels values. *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

Source data may be ***Constrained*** or ***Unconstrained***; the output *type* will match.

YCC

Parameters

luma: XieTypConstant
levels: XieTypLevels
scale: XieTypFloat
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Luma represents the proportions of red, green, and blue in luminance ***Y***. *Levels* determines the output if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Scale* is used to expand the source pixels (a typical value is about 1.35 to 1.4 in the literature). *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

Source data may be ***Constrained*** or ***Unconstrained***; the output *type* will match.

Convolution Edge Techniques

Various methods of handling edge conditions are provided for **Convolve**. These techniques determine what pixel values are used when **Convolve** requires data beyond the image bounds. **ConvolveTechnique**s only come into play when the kernel is positioned partially off the edge of the image. Data around the edges of a **ProcessDomain** are convolved with adjacent image pixels wherever possible.

Constant

Parameters

constant: XieTypConstant

Constant uses the value specified by *constant* if pixels are required from beyond the edge of the image.

Replicate

Parameters

< none >

Replicate uses the nearest edge pixel value if pixels are required from beyond the edge of the image.

Decode Techniques

UncompressedSingle

Parameters

fill-order: XieTypOrientation
pixel-order: XieTypOrientation
pixel-stride: CARD8
left-pad: CARD8
scanline-pad: CARD8

UncompressedTriple

Parameters

fill-order: XieTypOrientation
pixel-order: XieTypOrientation
band-order: XieTypOrientation
interleave: XieTypInterleave
pixel-stride: XieTypTripletofCARD8
left-pad: XieTypTripletofCARD8
scanline-pad: XieTypTripletofCARD8

These techniques are used when no compression scheme has been applied to the image data. They are used for ***SingleBand*** and ***TripleBand*** data, respectively. Their parameters define the format of the **DataStream** of uncompressed data (the server may reformat the data as it chooses prior to processing or storage).

The following parameters are common to both techniques:

- When multiple pixels are put in the same byte or a pixel spans multiple bytes, *fill-order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first.
- For pixels that span a byte boundary, *pixel-order* defines whether the most or least significant bits of the pixel are transported first within the **DataStream**.
- *Pixel-stride* is the number of bits between the start of consecutive pixels within a scanline. It must be at least enough bits to contain the number of input *levels*.
- *Left-pad* is the number of pad bits preceding the first image pixel in each scanline. If the server's **Alignment** attribute is ***Alignable*** or *pixel-stride* fits the definition