CONVERT

Converts an image to the specified image type and depth

Syntax

rc = IMGOP(task-id, 'CONVERT', type);

type

specifies the type of image to convert to:

'GRAY'

a monochrome (black and white) image

'CMAP'

a color-mapped image

'RGBA'

an RGB image

Type: Character

Details

CONVERT performs dithering, quantizing, and other operations in order to reduce an image to a simpler form. It can also create a two-color (black and white) RGB image by converting a monochrome image to an RGBA image. Images that are originally gray-scale or black and white cannot be colorized. CONVERT acts on the currently selected image or on a specified image.
Example

Convert an RGB image to a dithered monochrome image:

```
rc=imgop(task-id,'READ','rgb.tif');
rc=imgop(task-id,'CONVERT','GRAY');
rc=imgop(task-id,'WRITE','gray.tif');
```

Convert the GRAY image back to RGB. Because all color information is lost, the final RGB image has only two colors:

```
rc=imgop(task-id,'READ','gray.tif');
rc=imgop(task-id,'CONVERT','RGBA');
rc=imgop(task-id,'WRITE','rgb.tif');
```

COPY

Copies an image

Syntax

```
rc=IMGOP(task-id,'COPY',source-image-id
         <, destination-image-id>);
```

**source-image-id**

is the identifier of the image to copy.

Type: Numeric

**destination-image-id**

is the new identifier of the copied image.

Type: Numeric

Details

COPY copies an image from source-image-id to destination-image-id. That is, it assigns another image identifier to an image. If destination-image-id is not specified, it copies to the currently selected image. The copied image is not automatically displayed.

Example

Simulate zooming and unzooming an image:

```
path=lnamemk(5,'sashelp.imagapp.gkids','format=cat');
rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ_PASTE',1,1,path);
if (zoom eq 1) then
```
CREATE_IMAGE

Creates a new image that is stored in memory

---------

Syntax

rc=IMGOP(task-id,'CREATE_IMAGE',width,height,
    type, depth<,color-map-len>);

width

is the width of the new image in pixels.
    Type: Numeric

height

is the height of new image in pixels.
    Type: Numeric

type

is the type of the image. These values match the values that QUERYN returns for type:
1    specifies a GRAY image (1-bit depth)
2    specifies a CMAP image 8-bit depth)
3    specifies an RGB image (24-bit depth)
    Type: Numeric

depth

is the depth of the new image. In Version 7, the depth must match the value given for type, above.
    Type: Numeric

color-map-len

is the number of colors in the color map. This value is used only with a type of 2 (CMAP). If not specified, it defaults to 256.
Details
CREATE_IMAGE creates an “empty” image in which all data and color map values are set to 0 (black). You must use SET_COLORS to set the color map and SET_PIXEL to set the pixel values. Note that processing an entire image in this manner can be very slow.

Example
Copy an image. Note that the COPY command is a much faster way of doing this, and this example is here to show how to use the commands.

COPY:
width=0; height=0; type=0; depth=0; cmaplen=0;
r=0; g=0; b=0; pixel=0; pixel2=0; pixel3=0;

task-id=imginit(0,'nodisplay');
task-id2=imginit(0,'nodisplay');

/* read and query original image */
rc=imgop(task-id,'READ','first.tif');
rc=imgop(task-id,'QUERYN','WIDTH',width);
rc=imgop(task-id,'QUERYN','HEIGHT',height);
rc=imgop(task-id,'QUERYN','TYPE',type);
rc=imgop(task-id,'QUERYN','DEPTH',depth);
rc=imgop(task-id,'QUERYN','COLORMAP_LEN',
cmaplen);

/* Create the new image */
rc=imgop(task-id2,'CREATE_IMAGE',width,height,
  type,depth);

/* Copy the color map */
do i=0 to cmaplen-1;
  rc=imgop(task-id,'GET_COLORS',i,r,g,b);
  rc=imgop(task-id2,'SET_COLORS',i,r,g,b);
end;

/* Copy the pixels */
do h=0 to height-1;
do w=0 to width-1;
  rc=imgop(task-id,'GET_PIXEL',w,h,pixel,
    pixel2,pixel3);
  rc=imgop(task-id2,'SET_PIXEL',w,h,pixel,
    pixel2,pixel3);
end;
end;

/* Write out the new image */
rc=imgop(task-id2,'WRITE','second.tif',
  ‘format=tif’);
rc=imgterm(task-id);
rc=imgterm(task-id2);
return;
CROP

Crops the selected image

Syntax

\[
\text{rc}=\text{IMGOP}(\text{task-id},'CROP',\text{start-x},\text{start-y}, \\
\text{end-x}, \text{end-y}); \\
\text{region-id}=\text{PICFILL}(\text{graphenv-id}, \text{type,ulr,ulc,} \\
\text{lrr,lrc,source,}'CROP',<\text{arguments}>>); \\
\]

start-x

is the row number of the upper corner.
Type: Numeric

start-y

is the column number of the upper corner.
Type: Numeric

designates

is the row number of the lower corner.
Type: Numeric

designates

is the column number of the lower corner.
Type: Numeric

Details

The start-x, start-y, end-x, and end-y points use units of pixels and are included in the new image. The top left corner of the image is (0,0).

Example

Display an image and then crop it:

name=lnamemk(1,path);
rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ_PASTE',1,1,name);

if (crop eq 1) then do;
   rc=imgop(task-id,' CROP',ucx,ucy,lcx,lcy);
   rc=imgop(task-id,'PASTE',1,1);
end;
DESTROY

Removes an image from memory and from the display

Syntax
rc=IMGOP(task-id,'DESTROY',image-id);

image-id contains the identifier of the image to remove.
Type: Numeric

Details
DESTROY removes an image from memory and from the display. This command acts on the currently selected image unless image-id is specified. The command does not affect the image stored in the external file or catalog.

Example
Remove an image from the display:
if (remove=1 and imgnum > 0)
then
   rc=imgop(task-id,'DESTROY',imgnum);

DESTROY_ALL

Removes all images from memory and from the display

Syntax
rc=IMGOP(task-id,'DESTROY_ALL');

Details
DESTROY_ALL runs the DESTROY command for all images in memory. The external image files are not affected.

Example
Remove all images:
if (clear=1) then
    rc=imgop(task-id,'DESTROY_ALL');

DITHER

Dithers an image to a color map

Syntax
rc=IMGOP(task-id,'DITHER');
region-id=PICFILL(graphenv-id, type, ulr, ulc,
    lrr, lrc, source < 'DITHER', arguments >>);

Details
DITHER acts on the currently selected image. It dithers an image to the current color map: the one specified by a previous GENERATE_CMAP, STANDARD_CMAP, or GRAB_CMAP command.

Like the MAP_COLORS command, DITHER reduces the number of colors in an image. Unlike the MAP_COLORS command, DITHER attempts to choose colors by looking at pixels in groups, not as single pixels, and tries to choose groups that will result in the appropriate color. This is similar to the halftoning algorithm that print vendors use to show multiple colors with the use of only four ink colors. This command is much more computationally expensive than the other color-reduction commands, but it handles continuous-tone images much better.

Example
Dither an image:
if (dither=1) then
    do;
        rc=imgop(task-id,'GENERATE_CMAP','COLORRAMP',
            5, 5, 4);
        rc=imgop(task-id,'DITHER');
        rc=imgop(task-id,'PASTE');
    end;

DITHER_BW

Dithers the selected image to a monochrome black and white image

Syntax
rc=IMGOP(task-id,'DITHER_BW');
region-id=\texttt{PICFILL}(\texttt{graphenv-id}, \texttt{type}, \texttt{ulr}, \texttt{ulc},
  \texttt{lrr}, \texttt{lrc}, \texttt{source}, '<\texttt{DITHER\_BW}', \texttt{arguments}>>);

\textbf{Details}

This command reduces an image to a black-and-white image. \texttt{DITHER\_BW} is much more efficient for this task than the general purpose \texttt{DITHER} command.

\textbf{Example}

Dither an image either to black and white or to a color map:

\begin{verbatim}
if (dither=1) then
  do;
    rc=imgop(task-id,'DITHER\_BW');
    rc=imgop(task-id,'PASTE');
  end;
if (dither=2) then
  do;
    rc=imgop(task-id,'GENERATE\_CMAP',
     'COLORRAMP',5,5,4);
    rc=imgop(task-id,'DITHER');
    rc=imgop(task-id,'PASTE');
  end;
\end{verbatim}

\textbf{EXECLIST}

\textit{Executes a list of commands}

\textbf{Syntax}

\begin{verbatim}
rc=imgop(task-id,'EXECLIST',commandlist-id);
\end{verbatim}

\begin{verbatim}
region-id=\texttt{PICFILL}(\texttt{graphenv-id}, \texttt{type}, \texttt{ulr}, \texttt{ulc},
  \texttt{lrr}, \texttt{lrc}, \texttt{source}, '<EXECLIST>', \texttt{arguments}>>);
\end{verbatim}

\texttt{commandlist-id}

contains the identifier of the SCL list of commands to pass and execute. The commands are processed as the task starts. A value of zero means that no list is passed.

Type: Numeric

\textbf{Details}

\texttt{EXECLIST} provides a mechanism for sending multiple commands to be processed at one time. If your program includes the same set of commands several times, you can fill an SCL list with those commands and then use \texttt{EXECLIST} to execute the commands.
Example

Create an SCL list that consists of two sublists. Each sublist contains one item for a command name and one item for each command argument.

```scl
length rc 8;
init:
  task-id=imginit(0);
  main_list=makelist(0, 'G');

  sub_list1=makelist(0, 'G');
  main_list=setiteml(main_list, sub_list1, 1, 'Y');
  sub_list1=setitemc(sub_list1, 'WSIZE', 1, 'Y');
  sub_list1=setitemn(sub_list1, 500, 2, 'Y');
  sub_list1=setitemn(sub_list1, 500, 3, 'Y');

  sub_list2=makelist(0, 'G');
  main_list=setiteml(main_list, sub_list2, 2, 'Y');
  sub_list2=setitemc(sub_list2, 'WTITLE', 1, 'Y');
  sub_list2=setitemc(sub_list2, 'EXECLIST example',
                     2, 'Y');
  rc=imgop(task-id, 'EXECLIST', main_list);
return;

main:
return;
term:
  rc=imgterm(task-id);
return;
```

FILTER

Applies a filter to an image

Syntax

```
rc=IMGOP(task-id,'FILTER',filter-type, matrix);
```

filter-type

must be specified as ‘CONVOLUTION’. Other filter types will be added in the future.

  Type: Character

matrix

contains the matrix size, the filter matrix, the divisor, the bias, and 1 if you want to use the absolute value of the resulting value. If not specified, the defaults are 1 for divisor, 0 for bias and 0 for not using the absolute value. Separate each number with a space.

  Type: Character
Details

The FILTER command supports user-provided convolution filters. A filter matrix is moved along the pixels in an image, and a new pixel value is calculated and replaced at the pixel that is at the center point of the filter matrix. The new value is determined by weighting nearby pixels according to the values in the filter matrix.

A detailed explanation of the concept and theory behind filtering is beyond the scope of this document. However, it is explained in many textbooks. For example, see Digital Image Processing, by Rafael Gonzalez and Paul Wintz, and The Image Processing Handbook, by John C. Russ.

The equation used is shown in Figure A1.1 on page 750.

Figure A1.1 Calculating New Pixel Values

\[
N = \left( \left\{ \sum_{i=1}^{\text{matrix.size}} P_i M_i \right\} \right) / \text{Divisor} + \text{Bias}
\]

Where:
- \(N\) is the new pixel value (replaced in center of matrix).
- \(P\) is the pixel value in the matrix area.
- \(M\) is the filter matrix.
- Divisor is the divisor value provided.
- Bias is the bias value provided.
- Matrix.size is the size of the filter matrix (e.g. in a 3x3 filter, matrix.size is 9)

Example

<table>
<thead>
<tr>
<th>Image Pixels (P)</th>
<th>Filter Matrix (M)</th>
<th>Products</th>
<th>Sums</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 10 100</td>
<td>-1 -1 -1</td>
<td>25 10 100</td>
<td>-135</td>
</tr>
<tr>
<td>10 35 25</td>
<td>-1 9 -1</td>
<td>-10 315 -25</td>
<td>280</td>
</tr>
<tr>
<td>25 0 100</td>
<td>-1 -1 -1</td>
<td>-25 0 -100</td>
<td>-125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum of sums</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divisor</td>
<td>1</td>
</tr>
<tr>
<td>Bias</td>
<td>1</td>
</tr>
<tr>
<td>New Pixel (N)</td>
<td>21</td>
</tr>
</tbody>
</table>

Consider the following 3x3 matrix:

\[
\begin{pmatrix}
-1 & -2 & -3 \\
4 & 5 & 6 \\
-7 & 8 & -9
\end{pmatrix}
\]
Design the matrix with a divisor of 1 and a zero bias, and use the absolute value of the answer:

```
matrix="3 -1 -2 -3 4 5 6 -7 8 -9 1 0 1";
rc=imgop(tid,'FILTER','CONVOLUTION',matrix);
```

Note: Normally calculated values that are larger than 255 and smaller than zero are normalized to 255 or zero. If 1 is set for ‘absolute value’, then negative numbers are first converted to positive numbers.

A filter selection and creation window is available. An example of using it is in the image sample catalog (imagedmo) named FILTEXAM.FRAME. It is essentially the same window that is used in the Image Editor. It accesses the filters that are shipped with the Image Editor.

---

**GAMMA**

Applies a gamma value to the selected image

---

**Syntax**

```
rc=IMGOP(task-id,'GAMMA',gamma-value);
region-id=PICFILL(graphenv-id, type,ulr,ulc,lrr,lrc,source,<'GAMMA' <,arguments>>);
```

**gamma-value**

is the gamma value to apply to the image.

Type: Numeric

**Details**

GAMMA corrects the image by either darkening or lightening it. Gamma values must be positive, with the most useful values ranging between 0.5 and 3.0. A gamma value of 1.0 results in no change to the image. Values less than 1.0 darken the image, and values greater than 1.0 lighten it.

**Example**

Apply a gamma value that has previously been stored in GAMNUM:

```
if (gamma eq 1) then
do;
    rc=imgop(task-id,'GAMMA',gamnum);
    if (rc ne 0) then _msg_='gamma error';
    rc=imgop(task-id,'PASTE');
end;
```
GENERATE_CMAP

Generates a color map for the selected image

Syntax

rc=IMGOP(task-id,'GENERATE_CMAP', COLORRAMP,reds,greens, blues);
rc=IMGOP(task-id,'GENERATE_CMAP', GRAYRAMP,n);

reds
is the number of red colors to generate.
  Type: Numeric

greens
is the number of green colors to generate.
  Type: Numeric

blues
is the number of blue colors to generate.
  Type: Numeric

n
is the number of gray colors to generate.
  Type: Numeric

Details

GENERATE_CMAP generates two kinds of color maps:

COLORRAMP
is a color ramp of RGB colors that fill the RGB color spectrum, given the desired number of red, green, and blue shades to use. This command generates a color map of \( \text{reds} \times \text{greens} \times \text{blues} \) colors, with a maximum of 256 colors allowed. It is possible to generate a color map that consists only of reds, greens, or blues by specifying that only one shade be used for the other two colors.

GRAYRAMP
is a color map that consists only of grays. The number of shades of gray is limited to 256.

After the color map is generated, it can be applied to an image with either the DITHER command or the MAP_COLORS command.

Example

Use the GENERATE_CMAP command to generate a color ramp and a gray ramp, each containing 100 color map entries:
gray:
    rc==imgop(task-id,'GENERATE_CMAP','GRAYRAMP',100);
    return;

color:
    rc==imgop(task-id,'GENERATE_CMAP','COLORRAMP',5,5,4);
    return;

GET_BARCODE

Returns the value of the specified bar code

Syntax
rc=IMGOP(task-id,'GET_BARCODE',
    bar-code-type, return-string,<x1,y1,x2,y2>);

bar-code-type
    is a character string that contains one value from the following list:
    'CODE39'    code 39 bar codes
    'CODE39X'   extended code 39 bar codes
    'CODE128'   code 128 bar codes.
    Type: Character

return-string
    contains the returned value. Remember to make this variable long enough to hold
    the longest value that could be returned.
    Type: Character

x1,y2
    are the upper coordinates of the area in the image to search for the bar code. The
    default is 0,0.

x2,y2
    are the lower coordinates of the area in the image to search for the bar code. The
    default is the width and height of the image. Note that the area specified for the
    bar-code location can be larger than the bar code. This area should be relatively free
    of things like other text.

Details
Given an image with a bar code, the GET_BARCODE command attempts to decode the
bar code and then returns the value of the bar code. The bar code can be decoded only if
it is clear in the image. The DPI resolution used in scanning the image will determine
how clearly the bar code appears in the image. Below 200 DPI, recognition is very poor.
Example

Return the value of the bar code that is located in the 10,10,300,200 area of the image:

rc=imgop(taskid,'GET_BARCODE','CODE39',retstring,
10,10,300,200);

GET_COLOR

Returns the RGB values of the index positions of a color map for the selected image

Syntax

rc=IMGOP(task-id,'GET_COLOR',index,red,green,blue);

index
contains the identifier for the color map index.
  Type: Numeric

red
  is the red value for the index.
  Type: Numeric

green
  is the green value for the index.
  Type: Numeric

blue
  is the blue value for the index.
  Type: Numeric

Details

If index is outside the valid range for the color map, an error is returned. The color values are in the range of 0 to 255.

Example

See the example for “CREATE_IMAGE” on page 743.

GET_PIXEL

Returns the pixel value of a specified position in the selected image
Syntax
rc=IMGOP(task-id,‘GET_PIXEL’,x,y,red,<green, blue>);

x
is the row location in the image.
Type: Numeric

y
is the column location in the image.
Type: Numeric

red
is either the red value of an RGB image or the pixel value for a CMAP or GRAY image.
Type: Numeric

green
is the green value for an RGB image and is ignored for all others.
Type: Numeric

blue
is the blue value for an RGB image and is ignored for all others.
Type: Numeric

Details
An error is returned if any of the values are out of bounds. The colors for a CMAP and RGB image must be between 0 and 255. For a GRAY image, GET_PIXEL returns a red value of either 0 or 1.

Example
See the example for “CREATE_IMAGE” on page 743

GRAB_CMAP

Grabs the color map from the selected image

Syntax
rc=IMGOP(task-id,’GRAB_CMAP’);
Details

After the color map is grabbed, it can be applied to another image with either the DITHER command or the MAP_COLORS command.

Example

Grab the color map of one image and then apply it to another image with the DITHER command:

```plaintext
rc=imgop(task-id,'READ','image-1');
rc=imgop(task-id,'GRAB_CMAP');
rc=imgop(task-id,'READ','image-2');
rc=imgop(task-id,'DITHER');
```

MAP_COLORS

Maps colors to the closest color in the selected color map

Syntax

```plaintext
rc=IMGOP(task-id,'MAP_COLORS',option);
region-id=PICFILL(graphenv-id,type,ulr,ulc,lrr,lrc,source,'MAP_COLORS',<arguments>);
```

**option**

specifies the order in which the colors are to be mapped. By default, the colors are mapped in an order that is defined by an internal algorithm. Specify 'SAME_ORDER' to force the color map of the image to be in the same order as the selected color map.

Type: Character

Details

MAP_COLORS acts on the currently selected image. Like the DITHER and QUANTIZE commands, MAP_COLORS reduces the number of colors in a color image. Unlike DITHER, MAP_COLORS attempts to choose colors by looking at pixels individually, not in groups. This technique is much less computationally expensive than DITHER, although it does not handle continuous-tone images as well.

Continuous-tone images contain many shades of colors. Because MAP_COLORS maps the colors in an image to their closest colors in the color map, many of the shades of a color remap to the same color in the color map. This can reduce the detail in the image. For example, a continuous-tone, black-and-white image would contain several shades of gray in addition to black and white. When MAP_COLORS remaps the colors in the image, the shades of gray are mapped to either black or white, and much of the detail in the image is lost.

Unlike the QUANTIZE command, MAP_COLORS is passed a particular color map to use. Therefore, multiple images can be reduced to the same color map, further reducing the number of colors used in a frame that contains multiple images. The algorithm
Looks at each pixel in the image and determines the closest color in the color map. This type of algorithm works best for images that are not continuous-tone images, such as charts, cartoon images, and so on.

Specify the option ‘SAME_ORDER’ if you are mapping several images and you want the color map to be identical for all of them.

**Example**

Grab the color map of one image and then apply it to another image with the MAP_COLORS command:

```plaintext
rc=imgop(task-id,'READ',image1);
rc=imgop(task-id,'GRAB_CMAP');
rc=imgop(task-id,'READ',image2);
rc=imgop(task-id,'MAP_COLORS');
```

**MIRROR**

Mirrors an image

---

**Syntax**

```plaintext
rc=IMGOP(task-id,'MIRROR');
```

**Details**

MIRROR acts on the currently selected image. It flips an image on its vertical axis, resulting in a “mirror” copy of the original image.

**Example**

Mirror an image:

```plaintext
if (mirror=1) then
   rc=imgop(task-id,'MIRROR');
```

**NEGATE**

Changes an image to a negative

---

**Syntax**

```plaintext
rc=IMGOP(task-id,'NEGATE');
```
region-id=\textbf{PICFILL}(\texttt{graphenv-id,typ,ulr,ulc,lrr,lrc,source,\textless \texttt{NEGATE}\textgreater \texttt{,arguments\textgreater \textgreater});

**Details**

\textsc{NEGATE} acts on the currently selected image. It creates a photographic ‘negative’ of the image by reversing the use of dark/light colors. The negative is created by replacing each color with its complement.

**Example**

Create a negative of an image:

\begin{verbatim}
if (negative=1) then
  rc=imgop(task-id,'NEGATE');
\end{verbatim}

---

**PASTE**

Displays an image at a specified location

**Syntax**

\begin{verbatim}
rc=\textbf{IMGOP}(\texttt{task-id,'PASTE'\textless\texttt{x,y}\textgreater});
\end{verbatim}

\begin{itemize}
  \item \texttt{x}
    \begin{itemize}
      \item is the X coordinate of the top left corner of the image.
      \item Type: Numeric
    \end{itemize}
  \item \texttt{y}
    \begin{itemize}
      \item is the Y coordinate of the top left corner of the image.
      \item Type: Numeric
    \end{itemize}
\end{itemize}

**Details**

\textsc{PASTE} acts on the currently selected image. If no coordinates are specified, the selected image is displayed either at location 0,0 or at the coordinates set by a previous \textsc{PASTE}. To set new coordinates, you can use a \textsc{PASTE} command with no image specified. Coordinates that are specified by a new \textsc{PASTE} override previous settings.

**Example**

Display an image with its upper left corner at 200, 200:

\begin{verbatim}
if (display=1) then
  rc=imgop(task-id,'PASTE',200,200);
\end{verbatim}
**PASTE_AUTO**

Displays an image automatically

---

**Syntax**

\[ \text{rc} = \text{IMGOP}(\text{task-id},'PASTE\_AUTO',x,y); \]

\( x \)

is the X coordinate of the top left corner of the image.
- **Type:** Numeric

\( y \)

is the Y coordinate of the top left corner of the image.
- **Type:** Numeric

**Details**

`PASTE\_AUTO` acts on the currently selected image. It provides the same basic function as `PASTE`. In addition, `PASTE\_AUTO` modifies an image by dithering it or by reducing the number of colors it uses, so that you can display it on the current device. It also attempts to prevent switching to false colors or to a private color map.

**Example**

Automatically display an image with its upper left corner at 200, 200:

```plaintext```
if (display=1) then
    rc=imgop(task-id,'PASTE\_AUTO',200,200);
```

---

**QUANTIZE**

Reduces the number of colors used for an image

---

**Syntax**

\[ \text{rc} = \text{IMGOP}(\text{task-id},'QUANTIZE',\text{colors}); \]

\[ \text{region-id} = \text{PICFILL}(\text{graphenv-id},\text{type},\text{ulr},\text{ulc},\text{lrr},\text{lrc}, \text{source},'<QUANTIZE',\text{arguments}>); \]

**colors**

is the number of colors to use for the image. The `colors` variable must have a value from 2 through 256.
Type: Numeric

Details
QUANTIZE acts on the currently selected image. It generates a color-mapped image for which the command assigns the values in the color map. QUANTIZE results in a very good approximation of the image, with the possible negative effect that two or more images quantized to the same number of colors may still use different colors for each image. (The algorithm is an adaptation of the Xiaolin Wu algorithm, as described in Graphics Gems II.*)

Example
Reduce the number of colors for an image to the number stored in NUMCOLOR:

```c
if (quantize eq 1) then
  rc=imgop(task-id,'QUANTIZE',numcolor);
```

---

**QUERYC, QUERYL, and QUERYN**

Query information about images

**Syntax**

```c
rc=IMGOP(task-id,'QUERYC',attribute,information);
rc=IMGOP(task-id,'QUERYL',attribute,list-id);
rc=IMGOP(task-id,'QUERYN',attribute,information);
```

**attribute**
is the value to report. Attributes for QUERYC are listed in “Attributes for the QUERYC Command” on page 761. Attributes for QUERYL are listed in “Attributes for the QUERYL Command” on page 761. Attributes for QUERYN are listed in “Attributes for the QUERYN Command” on page 762.

Type: Character

**information**
contains the information that is returned by QUERYC and QUERYN. This variable is character when used by QUERYC and numeric when returned by QUERYN.

Type: Character or Numeric

**list-id**
contains the identifier for the SCL list of information items that are returned by QUERYL. See attribute for details.

---

Attributes for the QUERYC Command

The values for attribute for QUERYC are:

- **DESCRIPT**
  
  returns information about the image size and color map. The information can be up to 45 characters long.

- **FILENAME**
  
  returns the image path string.

- **FORMAT**
  
  returns the original file format, such as GIF.

- **TYPE**
  
  returns the IMAGE type, which can be 'CMAP', 'GRAY', or 'RGBA'.

Attributes for the QUERYL Command

The values for attribute for QUERYL are:

- **ACTIVE_LIST**
  
  returns an SCL list containing the identifiers for all active images (images that are being used but that are not necessarily visible).

- **VISIBLE_LIST**
  
  returns an SCL list containing the identifiers for all currently displayed images.

- **SELECT_INFO**
  
  returns a named SCL list containing the numeric values for the currently selected image:

  - **IS_ACTIVE**
    
    has a value of 1 if the image is being used and has data associated with it. If IS_ACTIVE=1, the following items are also returned:

    - **WIDTH** the image width in pixels
    - **HEIGHT** the image height in pixels
    - **DEPTH** the image depth
    - **TYPE** the image type: 'CMAP', 'GRAY', 'RGBA'

  - **IS_VISIBLE**
    
    has a value of 1 if the image is being displayed.

  - **XPOSN**
    
    is the x position.

  - **YPOSN**
    
    is the y position.

  - **NCOLORS**
    
    is the number of colors, if TYPE='CMAP' (color mapped)
RDEPTH
is the red depth, if TYPE='RGBA'

GDEPTH
is the green depth, if TYPE='RGBA'

BDEPTH
is the blue depth, if TYPE='RGBA'

ADEPTH
is the alpha depth (degree of transparency), if TYPE='RGBA'

GLOBAL_INFO
returns a named list that contains the following items:

NUM_ACTIVE
is the number of active images used but not necessarily visible.

SELECT
is the identifier of the currently selected image.

WSIZE_WIDTH
is the window width in pixels.

WSIZE_HEIGHT
is the window height in pixels.

Attributes for the QUERYN Command

The values for attribute for QUERYN are:

ADEPTH
returns the alpha depth.

BDEPTH
returns the blue depth.

COLORMAP-LEN
returns the size of the color map.

DEPTH
returns the image depth.

GDEPTH
returns the green depth.

HEIGHT
returns the image height in pixels.

IS_BLANK
returns a value that indicates whether the current page is blank:
1     blank
0     not blank (valid for monochrome images only).
NCOLORS
returns the number of colors.

RDEPTH
returns the red depth.

SELECT
returns the identifier of the currently selected image.

TYPE
returns the image type:
1    gray-scale
2    color mapped
3    RGBA.

WIDTH
returns the image width in pixels.

Details
The QUERYC, QUERYL, and QUERYN commands return information about all images as well as above the Image window. QUERYC returns the values of character attributes. QUERYL returns the values of attributes stored in an SCL list. QUERYN returns the values of numeric attributes. These commands act on the image that is currently selected.

Examples

Example 1: Using QUERYC    Display an image's description, filename, format, and type:

rc=imgop(task-id,'READ',
         '/usr/local/images/color/misc/canoe.gif');
rc=imgop(task-id,'QUERYC','DESCRIPT',idescr);
pout idescr=;
rc=imgop(task-id,'QUERYC','FILENAME',ifile);
pout ifile=;
rc=imgop(task-id,'QUERYC','FORMAT',iformat);
pout iformat=;
rc=imgop(task-id,'QUERYC','TYPE',itype);
pout itype=;

This program writes the following lines to the LOG window:

IDESCR=640x480 8-bit CMAP, 256 colormap entries
IFILE=/usr/local/images/color/misc/canoe.gif
IFORMAT=GIF
ITYPE=CMAP

Example 2: Using QUERYL
    Display the number of active images:

qlist=0;
rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ',path1);
rc=imgop(task-id,'SELECT',2);
rc=imgop(task-id,'READ',path2);
rc=imgop(task-id,'PASTE');
rc=imgop(task-id,'QUERYL','ACTIVE_LIST',qlist);
images=listlen(qlist);
put images=;

This program writes the following line to the LOG window:

images=2

Display an SCL list of information about the current image:

qlist=makelist();
rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ',path);
rc=imgop(task-id,'QUERYL','SELECT_INFO',qlist);
call putlist(qlist);

This program writes the following information to the LOG window:

(IS_ACTIVE=1 IS_VISIBLE=0 XPOSN=0 YPOSN=0 WIDTH=1024
HEIGHT=768 DEPTH=8 TYPE='CMAP' NCOLORS=253 )

Display an SCL list of information about the Image window:

qlist=makelist();
rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ',path);
rc=imgop(task-id,'QUERYL','GLOBAL_INFO',qlist);
call putlist(qlist);

When the program is run, the following lines are written to the LOG window:

(NUM_ACTIVE=1 SELECT=1 WSIZE_WIDTH=682
WSIZE_HEIGHT=475 )

Example 3: Using QUERYN  
Display information about the Image window. (Assume that all variables have been initialized prior to being used.)

rc=imgop(task-id,'READ',path);
rc=imgop(task-id,'QUERYN','SELECT',select);
rc=imgop(task-id,'QUERYN','HEIGHT',height);
rc=imgop(task-id,'QUERYN','WIDTH',width);
rc=imgop(task-id,'QUERYN','DEPTH',depth);
rc=imgop(task-id,'QUERYN','RDEPTH',rdepth);
rc=imgop(task-id,'QUERYN','GDEPTH',gdepth);
rc=imgop(task-id,'QUERYN','BDEPTH',bdepth);
rc=imgop(task-id,'QUERYN','ADEPTH',adepth);
rc=imgop(task-id,'QUERYN','NCOLORS',ncolors);
rc=imgop(task-id,'QUERYN','TYPE',type);
put select= height= width= depth= rdepth= gdepth=;
put bdepth= adepth= ncolors= type= ;

This program returns the following values:

SELECT=1 HEIGHT=470 WIDTH=625 DEPTH=8 RDEPTH=0
GDEPTH=0 BDEPTH=0 ADEPTH=0 NCOLORS=229 TYPE=2
READ

Reads an image from an external file, a SAS catalog, or a device

Syntax

rc=IMGOP(task-id, 'READ', pathname <,attributes >);
rc=IMGOP(task-id, 'READ', device-name, 'DEVICE=CAMERA | SCANNER <attributes >);

pathname

is the pathname of the external file that contains the image or the path string that is returned by the LNAMEMK function.

Type: Character
device-name

specifies the name of a camera or scanner:

'KODAKDC40'

Kodak DC 40 camera (available only under the Windows 95 operating system)

'HPSCAN'

HP Scanjet scanners (available only under Windows and HP/UX operating systems)

If you specify a device name, then you must use the DEVICE = attribute to indicate the type of device.

Type: Character

attributes

are file- or device-specific attributes. See "Attributes for Reading and Writing Files" on page 782 for possible choices.

Type: Character

Details

Read acts on the currently selected image. You can specify the file directly (using its physical filename path), or by using the information returned by a previous LNAMEMK function call. The LNAMEMK function creates a single character variable that contains location information about the image (even if it resides in a SAS catalog), as well as other image attributes.

The FORMAT = attribute must be specified for Targa images, for images that reside in SAS catalogs, and for host-specific formats. FORMAT is not required in other cases, but it is always more efficient to specify it.

Examples

- Read an image that is stored in a SAS catalog:

  path=lnamemk(5,'sashelp.imagapp.gfkids',
                  'format=cat');
Appendix 1

rc=imgop(task-id,'READ',path);

- Specify a file in the READ command:
  rc=imgop(task-id,'READ',
          '/usr/images/color/sign.gif');

- Read from a scanner:
  rc=imgop(task-id,'READ', 'hpscan',
           'device=scanner dpi=100');

- Take a picture with a camera:
  rc=imgop(task-id,'READ',
           'kodakdc40',
           'device=camera takepic');

- Read a Portable Networks Graphics image:
  rc=imgop(taskid,'READ','/images/test.png',
           'format=PNG ');

- Read an image using READ and wait 5 seconds before displaying the image after each PASTE command:
  rc=imgop(taskid,'READ',path);  
  rc=imgop(taskid,'PASTE');      
  rc=imgctrl(taskid,'WAIT',5);    
  rc=imgop(taskid,'READ',path2); 
  rc=imgop(taskid,'PASTE');      
  rc=imgctrl(taskid,'WAIT',5);

---

**READ_CLIPBOARD**

Reads an image from the host clipboard

---

**Syntax**

rc=IMGOP(task-id,'READ_CLIPBOARD');

**Details**

READ_CLIPBOARD acts on the currently selected image. On some hosts, the clipboard can be read only after you use the WRITE_CLIPBOARD command.

**Example**

Read an image from the clipboard and display it:

rc=imgop(task-id,'READ_CLIPBOARD');
rc=imgop(task-id,'PASTE');
READ_PASTE

Reads and displays an image

Syntax
rc=IMGOP(task-id,'READ_PASTE',x,y,image-path,<,attributes>);

x
is the X coordinate of the top left corner of the image.
Type: Numeric

y
is the Y coordinate of the top left corner of the image.
Type: Numeric

image-path
contains the pathname of the external file that contains the image or the path string that is returned by the LNAMEMK function.
Type: Character

attributes
are file-specific attributes. See “Attributes for Reading and Writing Files” on page 782 for possible choices.
Type: Character

Details
READ_PASTE acts on the currently selected image. It provides the same functionality as READ and PASTE. Notice that x and y are required.

Example
Read and paste an image that is stored in a SAS catalog:
path=lnamemk(5,'sashelp.imagapp.gfkids',
    'format=cat');
rc=imgop(task-id,'READ_PASTE',1,1,path);

READ_PASTE_AUTO

Reads and automatically displays an image
Syntax

\[ \text{rc} = \text{IMGOP}(\text{task-id},'\text{READ\_PASTE\_AUTO}',x,y,\text{image-path} <, \text{attributes}>); \]

\( x \)

is the X coordinate of the top left corner of the image.

Type: Numeric

\( y \)

is the Y coordinate of the top left corner of the image.

Type: Numeric

\( \text{image-path} \)

contains the pathname of the external file that contains the image or the path string that is returned by the \text{LNAMEMK} function.

Type: Character

\( \text{attributes} \)

are file-specific attributes. See “Attributes for Reading and Writing Files” on page 782 for possible choices.

Type: Character

Details

\text{READ\_PASTE\_AUTO} acts on the currently selected image. It provides the same functionality as \text{READ} and \text{PASTE\_AUTO}. Notice that \( x \) and \( y \) are required.

Example

Read and automatically paste an image that is stored in a SAS catalog:

\[ \text{path} = \text{lnamemk}(5,'\text{sashelp.imagapp.gfkids}','\text{format=cat}'); \]
\[ \text{rc} = \text{imgop}(\text{task-id},'\text{READ\_PASTE\_AUTO}',1,1,\text{path}); \]

\subsection*{ROTATE}

Rotates an image clockwise by 90, 180, or 270 degrees

Syntax

\[ \text{rc} = \text{IMGOP}(\text{task-id},'\text{ROTATE}',\text{degrees}); \]

\[ \text{region-id} = \text{PICFILL}(\text{graphenv-id},\text{type},\text{ulr},\text{ulc},\text{lrr},\text{lrc},\text{source},<\text{ROTATE}',<\text{arguments}>); \]
**degrees**

is the number of degrees to rotate the image: 90, 180, or 270.

Type: Numeric

**Details**

**ROTATE** acts on the currently selected image.

**Example**

Rotate an image the number of degrees stored in RV:

```plaintext
main:
    rc=imgop(task-id,'READ',path);
    if (rv ge 90) then
do;
        rc=imgop(task-id,'ROTATE',rv);
        rc=imgop(task-id,'PASTE');
    end;
    return;
```

---

**SCALE**

Scales an image

**Syntax**

```plaintext
rc=IMGOP(task-id,'SCALE',width,height<,algorithm>);
region-id=PICFILL(graphenv-id,type,ulr,ulc,lrr,lrc,source,<SCALE'<,arguments>>);
```

**width**

is the new width of the image (in pixels).

Type: Numeric

**height**

is the new height of the image (in pixels).

Type: Numeric

**algorithm**

specifies which scaling algorithm to use:

**BILINEAR**

computes each new pixel in the final image by averaging four pixels in the source image and using that value. The **BILINEAR** algorithm is more computationally expensive than **LINEAR**, but it preserves details in the image better.
LINEAR replicates pixels when the image is scaled up and discards pixels when the image is scaled down. The LINEAR algorithm yields good results on most images. However, it does not work very well when you are scaling down an image that contains small, but important, features such as lines that are only one pixel wide. LINEAR is the default.

Type: Character

Details
SCALE acts on the currently selected image. It scales the image to a new image. The size of the new image is specified in pixels; however, if one of the two values is -1, then the value used for that scale factor is computed to conserve the original image's aspect ratio.

Example
Double the size of an image:

main:
   rc=imgop(task-id,'READ',path);
   rc=imgop(task-id,'QUERYN','WIDTH',width);
   rc=imgop(task-id,'SCALE',2*width,-1);
   rc=imgop(task-id,'PASTE');
   return;

SELECT
Selects the image identifier to be used in other commands

Syntax
rc=IMGOP(task-id,'SELECT'<,image-id>);

image-id contains the identifier of the image to select. The default is 1. The image-id variable must be a number between 1 and 999, inclusive. Using lower sequential numbers (under 32) is more efficient.

Type: Numeric

Details
The main purpose of the SELECT command is to enable you to work with more than one image. The command specifies the image identifier to be used in all subsequent commands until another SELECT command is issued.

Only the COPY, DESTROY, and UNPASTE commands can act on either the currently selected image or on a specified image identifier.
Example

Display two images at once:

```c
rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ_PASTE',1,1,path1);
rc=imgop(task-id,'SELECT',2);
rc=imgop(task-id,'READ_PASTE',200,200,path2);
```

---

**SET_COLORS**

Assigns the RGB values for the index positions of a color map for the current image

**Syntax**

```c
rc=IMGOP(task-id,'SET_COLORS',index,red,green,blue);
```

- **index**
  - contains the identifier for the color map index.
    - Type: Numeric
- **red**
  - is the red value for the index.
    - Type: Numeric
- **green**
  - is the green value for the index.
    - Type: Numeric
- **blue**
  - is the blue value for the index.
    - Type: Numeric

**Details**

SET_COLORS acts on the currently selected image. It can be used with either a new image or an existing image. If index is outside the valid range for the color map an error is returned. The color values must be between 0 and 255.

**Example**

See the example for “CREATE_IMAGE” on page 743.

---

**SET_PIXEL**

Assigns the pixel value in an image at the specified position
Syntax
rc=IMGOP(task-id,'SET_PIXEL',x,y,red,<green, blue>);

x
is the row location in the image.
Type: Numeric

y
is the column location in the image.
Type: Numeric

red
is either the red value of an RGB image or the pixel value for a CMAP or GRAY image.
Type: Numeric

green
is the green value for an RGB image and is ignored for all others.
Type: Numeric

blue
is the blue value for an RGB image and is ignored for all others.
Type: Numeric

Details
SET_PIXEL acts on the currently selected image. It can be used with either a new image or an existing image. An error is returned if any of the values are out of bounds. The colors for a CMAP and an RGB image must be between 0 and 255. For a GRAY image, SET_PIXEL returns either 0 or 1 for red.

CAUTION:
Image data can be destroyed. Use this function carefully, or you can destroy your image data. SET_PIXEL overwrites the image data in memory and thus destroys the original image.

Example
See the example for “CREATE_IMAGE” on page 743.

STANDARD_CMAP
Selects a color map
Syntax

```plaintext
rc=IMGOP(task-id,'STANDARD_CMAP',color-map);
```

color-map

is the color map to designate as the current color map.

**BEST**

is a special, dynamic color map that can contain up to 129 colors. The color map contains the 16 personal computer colors, a set of grays, and an even distribution of colors. The colors are dynamically selected based on the capabilities of the display and on the number of available colors. The best set of colors is chosen accordingly.

**COLORMIX_CGA**

is the 16-color personal computer color map.

**COLORMIX_192**

is a 192-color blend.

**DEFAULT**

is an initial set of colors that is chosen by default. The available colors may vary between releases.

**SYSTEM**

is the color map for the currently installed device or system. The color map obtained is a “snapshot” of the color map for the current device and does not change as the device’s color map changes.

*Type: Character*

Details

**STANDARD_CMAP** specifies that the current color map should be filled with one of the “standard” image color maps. This new color map can be applied to any image by using either the DITHER command or the MAP_COLORS command.

**Example**

Select a new color map and use the DITHER command to apply it to an image:

```plaintext
rc=imgop(task-id,'STANDARD_CMAP','COLORMIX_CGA');
rc=imgop(task-id,'READ',path);
rc=imgop(task-id,'DITHER');
```

**THRESHOLD**

Converts color images to black and white using a *threshold* value
Syntax

rc=IMGOP(task-id,THRESHOLD’,value);

value

is a threshold value for converting standard RGB values to monochrome. Value can be:

1...255 sets the threshold that determines whether a color maps to black or white
0 defaults to 128
-1 calculates the threshold value by averaging all pixels in the image.

Type: Numeric

Details

The THRESHOLD command acts on the specified or currently selected image. It enables documents that are scanned in color to be converted to monochrome for applying optical character recognition (OCR) and for other purposes. Dithering is not a good technique for converting images when OCR is used.

The threshold is a color value that acts as a cut-off point for converting colors to black and white. All colors greater than the value map to white and all colors less than or equal to the value map to black.

The algorithm weights the RGB values using standard intensity calculations for converting color to gray scale.

TILE

Replicates the current image into a new image

Syntax

rc=IMGOP(task-id,’TILE’,new-width,new-height);

new-width

is the width (in pixels) for the tiled images to fill.

Type: Numeric

new-height

is the height (in pixels) for the tiled images to fill.

Type: Numeric
Details

TILE acts on the currently selected image. The size, in pixels, of the area for the new tiled image is specified by the two parameters new-width and new-height. The area defined by new-width x new-height is filled beginning in the upper left corner. The current image is placed there. Copies of the current image are added to the right until the row is filled. This process then starts over on the next row until the area defined by new-width x new-height is filled. For example, if the current image is 40 x 40 and new-width x new-height is 200 x 140, then the current image is replicated 5 times in width and 3.5 times in height. This technique is useful for creating tiled backdrops.

Note: Before tiling an image, you must turn off the SCALE option for the image.

Example

Create a 480 x 480 tiled image from a 48 x 48 image:

rc=imgop(task-id,'READ','sashelp.c0c0c.access',
    'format=cat');
rc=imgop(task-id,'TILE',480,480);

UNPASTE

Removes an image from the display

Syntax

rc=IMGOP(task-id,'UNPASTE'<,image-id>);

image-id
contains the identifier of the image to remove from the display.
Type: Numeric

Details

UNPASTE acts on the specified or currently selected image. It removes from the display the currently selected image or the image whose identifier is specified. The image is not removed from memory. UNPASTE enables you to remove an image from the display and to later repaste it without re-reading it.

Example

Display two images at once and then remove one of them:

rc=imgop(task-id,'SELECT',1);
rc=imgop(task-id,'READ_PASTE',1,1,name1);
rc=imgop(task-id,'SELECT',2);
rc=imgop(task-id,'READ_PASTE',200,200,name2);
...more SCL statements...
if (omit=1) then
    rc=imgop(task-id,'UNPASTE',1);
attributes

lists attributes that are specific to the file type. See “Attributes for Reading and Writing Files” on page 782.

Type: Character

Details

WRITE writes the currently selected image to an external file. The file can be specified either directly (using its physical filename path) or by using the information that was returned by a previous LNAMEMK function call. The LNAMEMK function creates a character variable that contains location information about the location of the image (even if it is to reside in a SAS catalog), as well as information about other image attributes.

The FORMAT= attribute must be specified if image-path does not include that information.

Examples

- Write an image to a SAS catalog:
  ```
  path=lnamemk
  (5,'mine.images.sign','FORMAT=CAT');
  rc=imgop(task-id,'WRITE',path);
  ```
- Specify a file in the WRITE command. (Notice that file attributes are included.)
  ```
  rc=imgop(task-id,'WRITE','/user/images/sign.tif',
           'FORMAT=TIFF COMPRESS=G3FAX');
  ```

WRITE_CLIPBOARD

Writes an image to the host clipboard

Syntax

```
rc=IMGOP(task-id,'WRITE_CLIPBOARD');
```  

Details

WRITE_CLIPBOARD acts on the currently selected image. The image must be pasted before it can be written to the system clipboard.

Example

Read in an image and write it to the clipboard:

```
rc=imgop
(task-id,'READ',path);
rc=imgop(task-id,'WRITE_CLIPBOARD');
```
WSIZE

Sets the size of the Image window

Syntax

\[ \text{rc} = \text{IMGCTRL}(\text{task-id}, \text{WSIZE}, \text{width}, \text{height} <, \text{xposition}, \text{yposition}>); \]

**width**

is the width of the window (in pixels).
Type: Numeric

**height**

is the height of the window (in pixels).
Type: Numeric

**xposition**

is the X coordinate of the top left corner.
Type: Numeric

**yposition**

is the Y coordinate of the top left corner.
Type: Numeric

Details

WSIZE sets the size of the Image window. Optionally, it positions the window at xposition and yposition. Some window managers may not support positioning.

Example

Make the Image window match the size of the image that is being displayed:

```python
main:
  height=0;
  width=0;
  rc=imgop(task-id,'READ',path);
  rc=imgop(task-id,'QUERYN','WIDTH',iwidth);
  rc=imgop(task-id,'QUERYN','HEIGHT',iheight);
  rc=imgctrl(task-id,'WSIZE',iwidth,iheight);
  rc=imgop(task-id,'PASTE',1,1);
return;
```

WTITLE

Specifies a title for the Image window
Syntax

```latex
rc=IMGCTRL(task-id,'WTITLE','title');
```

**title**

is the text to display as the window title.

Type: Character

Details

The specified title appears in parentheses after SAS: IMAGE in the title bar of the window.

Example

```latex
path=l namemk(5,catname,'format=cat');
rc=lnameget(path,type,name,form);
gname=scan(name,3,'.');
rc=imgctrl(tid,'wtitle',gname);
```