Listing of Digital Micrograph Scripting Commands (courtesy of Gatan inc.)

This listing originally appeared on Gatan inc's website. It was removed around 2004 when the built-in help reference was added to Digital Micrograph. It may not contain any recently added commands. It is provided here as a quick look-up reference for Digital Micrograph scripters. Simply open in a web browser or Word and search for commands or keywords of interest using CONTROL + f (PC) or COMMAND + f (Mac).

Notes About Parameters

1. Rectangles are passed as a quadruple (top, left, bottom, right).
2. Coordinate system transformations are passed as a quadruple (ox,oy,sx,sy), where (x,y) is transformed to (ox + sx*x, oy + sy * y).
3. An encoding parameter is used in several functions that take or return text. It is used to determine how to convert strings internally stored as unicode into strings usable in the script language. Currently, only the paramters 0, referring to the local character set, and 1, referring to the western encoding appropriate for the platform (Roman script on Macintosh, and code page 1252 on Windows), are allowed.

Script Classes

Other Script Functions

Class Component

Return to Top

Overview

Objects of type 'Component' are image document components, which include annotations (text, box, line, etc.), image displays, masks, and regions of interest. Components may be placed on a page, in an image display, or in a group, each of which is represented by a type of component. The containing component is referred to as the parent component, and it determines the coordinate system in which child locations and sizes are specified (the coordinate system of a container is its **child** coordinate system, and the child's **local** coordinate system). Other useful coordinate systems are

- **window** coordinates, whose units are screen pixels and whose origin is at the top-left corner of a window
- **page** coordinates, which is the coordinate system of the printed page. For the physical interpretation of these coordinates see the documentation for **ImageDocument** .
- **view** coordinates, which are the coordinates for the current view of the image document containing the component. These coordinates are used in several of **ImageDocument**'s methods

If a method requires or returns coordinates, they are usually specified in the component's **local** coordinate system. However, if the method ends in '**`InView`**' the coordinates are in **view** coordinates. Note that none of the methods that set coordinates use **view** coordinates. Thus, to manipulate the view of a component, use the methods ending in '**`InView`**' to get coordinates, and ImageDocument methods such as **`ImageDocumentSetRectInView`** to position the component.

The annotation type, used by the function **`NewAnnotation`** and the method **`ComponentGetType`**, can be one of the following numbers

2 - line annotation
3 - arrow annotation
4 - double arrow annotation
5 - box annotation
6 - oval annotation
8 - spot mask annotation
9 - array mask annotation
13 - text annotation
15 - bandpass mask annotation
17 - group annotation
19 - wedge mask annotation

The control points referred to in several of the methods can be one of

1 - start point of line
2 - end point of line
3 - position, used to move annotation (the position may be any point in the annotation, so it is only good for moving)
4 - top-left of rectangular annotations, one of the bandpass ovals
5 - top-right of rectangular annotations, one of the bandpass ovals
6 - bottom-left of rectangular annotations, one of the bandpass ovals
7 - bottom-right of rectangular annotations, one of the bandpass ovals
8 - other spot of spot mask
9 - one vector of wedge and array masks
10 - other vector of wedge and array masks
11 - top-left of other bandpass oval
12 - top-right of other bandpass oval

13 - bottom-left of other bandpass oval
14 - bottom-right of other bnadpass oval
15 - one spot of spot mask

Methods

`void` **`ComponentAddChildAfter`**`( Component` *`parent`*`, Component` *`child`*`, Component` *`annot_pos`* `)`
Adds 'child' to 'parent's list of sub-annotations after 'annot_pos'.
¬†

`void` **`ComponentAddChildAtBeginning`**`( Component` *`parent`*`, Component` *`child`* `)`
Adds 'child' to the beginning of 'parent's list of sub-annotations.
¬†

`void` **`ComponentAddChildAtEnd`**`( Component` *`parent`*`, Component` *`child`* `)`
Adds 'child' to the end of 'parent's list of sub-annotations.
¬†

`void` **`ComponentAddChildBefore`**`( Component` *`parent`*`, Component` *`child`*`, Component` *`annot_pos`* `)`
Adds 'child' to 'parent's list of sub-annotations before 'annot_pos'.
¬†

`Component` **`ComponentAddNewComponent`**`( Component` *`parent`*`, Number` *`type`*`, Number` *`f1`*`, Number` *`f2`*`, Number` *`f3`*`, Number` *`f4`* `)`
Creates a new annotaiton of type 'type' and adds it to 'parent'
¬†

`Component` **`ComponentClone`**`( Component` *`comp`*`, Boolean` *`doDeepCopy`* `)`
Returns a identical copy of the component and all its sub-components, copying associated images if 'doDeepCopy' is true.
¬†

`Number` **`ComponentCountChildren`**`( Component` *`comp`* `)`
Returns the number of sub components.
¬†

`Number` **`ComponentCountChildrenOfType`**`( Component` *`comp`*`, Number` *`type`* `)`
Returns the number of sub-components of type 'type'.
¬†

`void` **`ComponentGetBoundingRect`**`( Component` *`comp`*`, NumberVariable` *`t`*`, NumberVariable` *`l`*`, NumberVariable` *`b`*`, NumberVariable` *`r`* `)`
Gets the bounding rect of the annotation.
¬†

`void` **`ComponentGetBoundingRectInView`**`( Component` *`comp`*`, NumberVariable` *`t`*`, NumberVariable` *`l`*`, NumberVariable` *`b`*`, NumberVariable` *`r`* `)`
Gets the bounding rect of the annotation.
¬†

`Component` **`ComponentGetChild`**`( Component` *`comp`*`, Number` *`index`* `)`
Returns the 'index'th sub-component of 'comp'.
¬†

`Component` **`ComponentGetChildByID`**`( Component` *`comp`*`, Number` *`ID`* `)`
Returns the component child of 'comp' with id 'ID'.
¬†

`void` **`ComponentGetChildToLocalTransform`**`( Component` *`comp`*`, NumberVariable` *`off_x`*`, NumberVariable` *`off_y`*`, NumberVariable` *`scale_x`*`, NumberVariable` *`scale_y`* `)`
Gets the transformation from child to local coordinates.
¬†

`void` **`ComponentGetChildToPageTransform`**`( Component` *`comp`*`, NumberVariable` *`off_x`*`, NumberVariable` *`off_y`*`, NumberVariable` *`scale_x`*`, NumberVariable` *`scale_y`* `)`
Gets the transformation from child to page coordinates.
¬†

`void` **`ComponentGetChildToViewTransform`**`( Component` *`comp`*`, NumberVariable` *`off_x`*`, NumberVariable` *`off_y`*`, NumberVariable` *`scale_x`*`, NumberVariable` *`scale_y`* `)`
Gets the transformation from child to view coordinates.
¬†

`void` **`ComponentGetChildToWindowTransform`**`( Component` *`comp`*`, NumberVariable` *`off_x`*`, NumberVariable` *`off_y`*`, NumberVariable` *`scale_x`*`, NumberVariable` *`scale_y`* `)`
Gets the transformation from child to window coordinates.
¬†

`Boolean` **`ComponentGetControlPoint`**`( Component` *`comp`*`, Number` *`loc`*`, NumberVariable` *`x`*`, NumberVariable` *`y`* `)`
Returns the value '(x,y)' associated with the control point, and returns 'true' if the control point is valid
¬†

Component **ComponentGetDescendentByID**(Component *comp*, Number *ID*)
Returns the component child of 'comp' with id 'ID'.
¬†
Number **ComponentGetDrawingMode**(Component *comp*)
Gets the drawing mode of the image doucment component.
¬†
Number **ComponentGetFillMode**(Component *comp*)
Gets the fill mode of the image doucment component.
¬†
Number **ComponentGetFontAttributes**(Component *comp*)
Gets the attributes of the component's font.
¬†
String **ComponentGetFontFaceName**(Component *comp*)
Gets the face name of the component's font.
¬†
void **ComponentGetFontInfo**(Component *comp*, String *faceName*, NumberVariable *attributes*, NumberVariable *size*, NumberVariable *text_encoding*)
Gets a description of the component's font.
¬†
void **ComponentGetFontInfo**(Component *comp*, String *faceName*, NumberVariable *attributes*, NumberVariable *size*)
Gets a description of the component's font.
¬†
Number **ComponentGetFontSize**(Component *comp*)
Gets the point size of the component's font.
¬†
Number **ComponentGetID**(Component *annot*)
Gets the unique identifier of the annotation in the image document.
¬†
ImageDocument **ComponentGetImageDocument**(Component *annot*)
Gets the image document associated with the annotation.
¬†
void **ComponentGetLocalToPageTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from local to page coordinates.
¬†
void **ComponentGetLocalToViewTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from local to view coordinates.
¬†
void **ComponentGetLocalToWindowTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from local to window coordinates.
¬†
Component **ComponentGetNthChildOfType**(Component *comp*, Number *type*, Number *index*)
Returns the nth sub-component of type 'type'.
¬†
Component **ComponentGetParentComponent**(Component *comp*)
Gets the parent component of 'comp', if any.
¬†
ImageDisplay **ComponentGetParentImageDisplay**(Component *comp*)
Gets the parent image display of the 'comp', if any.
¬†
void **ComponentGetRect**(Component *comp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the rectangle of the annotation.
¬†
void **ComponentGetRectInView**(Component *comp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the rectangle of the annotation.
¬†
TagGroup **ComponentGetTagGroup**(Component *annot*)
Gets the tag group associated with the annotation.
¬†
Number **ComponentGetType**(Component *annot*)
Gets the type of the annotation.

¬†

Boolean **ComponentIsOfType**(Component *annot*, Number *type*)

Gets the type of the annotation.

¬†

Boolean **ComponentIsSelected**(Component *comp*)

Returns whether the component is selected.

¬†

Boolean **ComponentIsValid**(Component *annot*)

Returns true if 'annot' points to a valid object.

¬†

void **ComponentOffsetControlPoint**(Component *comp*, Number *loc*, Number *x*, Number *y*, Number *restrict_style*)

Changes the control point 'loc' of 'comp' by '(x,y)' using restrictions specified by 'restrict_style'.

¬†

void **ComponentPositionAroundPoint**(Component *comp*, Number *new_x*, Number *new_y*, Number *rel_x*, Number *rel_y*, Boolean *horz*, Boolean *vert*)

Moves the annotation so if 'horz', the 'rel_x' horizontal point in the bounding rect is at 'new_x', and if 'vert', the 'rel_y' vertical point in the bounding rect is at 'new_y'

¬†

void **ComponentRemoveFromParent**(Component *comp*)

Removes the image document component from its parent.

¬†

void **ComponentSetControlPoint**(Component *comp*, Number *loc*, Number *x*, Number *y*, Number *restrict_style*)

Sets the control point 'loc' of 'comp' to '(x,y)' using restrictions specified by 'restrict_style'.

¬†

void **ComponentSetDrawingMode**(Component *comp*, Number *mode*)

Sets the drawing mode of the image document component.

¬†

void **ComponentSetFillMode**(Component *comp*, Number *mode*)

Sets the fill mode of the image document component.

¬†

void **ComponentSetFontAttributes**(Component *comp*, Number *attributes*)

Sets the attributes of the component's font.

¬†

void **ComponentSetFontFaceName**(Component *comp*, String *face_name*)

Sets the face name of the component's font.

¬†

void **ComponentSetFontInfo**(Component *comp*, String *face_name*, Number *attributes*, Number *size*, Number *text_encoding*)

Sets the font information of the component's font.

¬†

void **ComponentSetFontInfo**(Component *comp*, String *face_name*, Number *attributes*, Number *size*)

Sets the font information of the component's font.

¬†

void **ComponentSetFontSize**(Component *comp*, Number *size*)

Sets the point size of the component's font.

¬†

void **ComponentSetRect**(Component *comp*, Number *top*, Number *left*, Number *bottom*, Number *right*)

Sets the rectangle of the annotation.

¬†

void **ComponentSetSelected**(Component *comp*, Boolean *select*)

Sets the selection status of the component.

¬†

void **ComponentTransformCoordinates**(Component *comp*, Number *off_x*, Number *off_y*, Number *scale_x*, Number *scale_y*)

Transforms the component by the specified transform.

¬†

Related Functions

Component **=**(Component, !)

¬†

Component **=**(Component, Component)

¬†
Component **ComponentNullify**(!)
¬†
void **GroupAnnotationUngroup**(Component *comp*)
Ungroups the group annotation.
¬†
Component **ImageDocumentGetComponentByID**(ImageDocument *imgDoc*, Number *id*)
Returns an annotation contained in this image document by id.
¬†
Component **ImageDocumentGetRootComponent**(ImageDocument *imgDoc*)
Gets the root annotation of the image document.
¬†
Component **NewArrowAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new arrow annotation.
¬†
Component **NewBoxAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new box annotation.
¬†
Component **NewComponent**(Number *type*, Number *f1*, Number *f2*, Number *f3*, Number *f4*)
Creates a new annotaiton of type 'type'
¬†
Component **NewDoubleArrowAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new double arrow annotation.
¬†
Component **NewGroupAnnotation**( )
Creates a new group annotation.
¬†
Component **NewLineAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new line annotation.
¬†
Component **NewOvalAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new oval annotation.
¬†
Component **NewPictureAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*, Number *picture*)
Creates a new picture annotation.
¬†
Component **NewTextAnnotation**(Number *left*, Number *top*, String *text*, Number *size*)
Creates a new text annotation.
¬†
void **PictureAnnotationSetPicture**(Component *comp*, Number *picture*)
Sets the picture of an annotation.
¬†
Number **TextAnnotationGetAlignment**(Component *comp*)
Gets the alignment of the text in the text annotation.
¬†
void **TextAnnotationGetFixedPoint**(Component *comp*, NumberVariable *x*, NumberVariable *y*)
Gets the fixed point of the text annotation.
¬†
Number **TextAnnotationGetResizeStyle**(Component *comp*)
Gets the resize style of the text annotation.
¬†
String **TextAnnotationGetText**(Component *comp*)
Gets the text of a text annotation.
¬†
void **TextAnnotationSetAlignment**(Component *comp*, Number *alignment*)
Sets the alignment of the text in the text annotation.
¬†
void **TextAnnotationSetFixedPoint**(Component *comp*, Number *x*, Number *y*)
Sets the fixed point of the text annotation.
¬†
void **TextAnnotationSetResizeStyle**(Component *comp*, Number *style*)
Sets the resize style of the text annotation.
¬†
void **TextAnnotationSetText**(Component *comp*, String *text*)

Sets the text of a text annotation.
¬†

Class ImageDisplay

Return to Top

Overview

Image displays are components used to display the contents of images. Component methods may be used to position these on the page, but specialized methods such as **`ImageDisplaySetImageRect`** are supplied to allow the position and size of the image portion of the display (excluding borders, captions, etc) to be specified. As with other methods dealing with coordinates, use those ending in **`'InView`**' to find the coordinates of the display as seen in a window. Currently only raster and rgb image displays have a well-defined coordinate system for child components, and this coordinate system has its origin at the top-left of the displayed image, and units equivalent to image pixels.

Image display types may be one of the following

-2 - Best image display. This is only used when setting a display type, and it produces line plots for 1-d images, and raster or rgb displays for 2-d data.
1 - Raster image display
2 - Surface plot image display
3 - Line plot image display
4 - RGB image display
7 - Spreadsheet image display
8 - Surface model image display (currently only available on the Macintosh).

Methods

`void` **`ImageDisplayAccumulateROIsToMask`**`( ImageDisplay` *`imgDisp`*`, ImageReference` *`mask`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*`, Number` *`mask_val`* `)`
Sets mask to mask_val at points in imageDisplay's rois
¬†

`void` **`ImageDisplayAddKeyListener`**`( ImageDisplay` *`imgDisp`*`, String` *`listener_key`*`, String` *`listener_script`* `)`
Adds the listener_script to the key listener list under the tag listener_key.
¬†

`void` **`ImageDisplayAddROI`**`( ImageDisplay` *`imgDisp`*`, ROI` *`roi`* `)`
Adds the roi to this image display.
¬†

`void` **`ImageDisplayChangeDisplayType`**`( ImageDisplay` *`imgDisp`*`, Number` *`new_type`* `)`
Changes the type of the image display.
¬†

`Number` **`ImageDisplayCountROIs`**`( ImageDisplay` *`imgDisp`* `)`
Returns the number of ROIs on this image display.
¬†

`void` **`ImageDisplayDeleteROI`**`( ImageDisplay` *`imgDisp`*`, ROI` *`roi`* `)`
Deletes the roi from this image display.
¬†

`Boolean` **`ImageDisplayDoesROIExist`**`( ImageDisplay` *`imgDisp`*`, String` *`name`* `)`
Determines whether the given ROI exists on this image display.
¬†

`void` **`ImageDisplayExportToFile`**`( ImageDisplay` *`imgDisp`*`, String` *`format`*`, String` *`file_name`* `)`
Exports the display to the file 'file_name' using the format 'format'.

¬†
ImageReference **ImageDisplayGetBufferedImage**(ImageDisplay *imgDisp*)
Gets the image resulting from the contrast transformation.
¬†
Number **ImageDisplayGetComplexMode**(ImageDisplay *imgDisp*)
Gets the complex mode of the display.
¬†
Number **ImageDisplayGetComplexModeRange**(ImageDisplay *imgDisp*)
Gets the parameter used in converting complex to real.
¬†
void **ImageDisplayGetContrastLimits**(ImageDisplay *imgDisp*, NumberVariable *low*, NumberVariable *high*)
Gets the contrast limits of the display.
¬†
Number **ImageDisplayGetContrastMode**(ImageDisplay *imgDisp*)
Returns the contrast mode.
¬†
void **ImageDisplayGetContrastParameters**(ImageDisplay *imgDisp*, NumberVariable *bright*, NumberVariable *contrast*)
Gets the parameters for the contrast mode.
¬†
ImageReference **ImageDisplayGetDisplayedImage**(ImageDisplay *imgDisp*)
Gets the image that is actually displayed in the image display.
¬†
void **ImageDisplayGetDisplayedLayers**(ImageDisplay *imgDisp*, NumberVariable *start*, NumberVariable *end*)
Gets the layers that are summed into the display.
¬†
Number **ImageDisplayGetDisplayType**(ImageDisplay *imgDisp*)
Returns type of the image display.
¬†
Boolean **ImageDisplayGetDoAutoSurvey**(ImageDisplay *imgDisp*)
Determines whether min and max are determined automatically.
¬†
ImageReference **ImageDisplayGetExportImage**(ImageDisplay *imgDisp*, Number *mode*, ImageDisplay *clut_display*)
Gets the image representation of the image as it appears on the screen at full resolution.
¬†
ImageReference **ImageDisplayGetImage**(ImageDisplay *imgDisp*)
Returns the single image displayed by the image display.
¬†
void **ImageDisplayGetImageAdjustRect**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Returns the image display outside the image rect
¬†
void **ImageDisplayGetImageAdjustRectInView**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Returns the image display outside the image rect in view coordinates
¬†
void **ImageDisplayGetImageRect**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the bounds of the image in the image display.
¬†
void **ImageDisplayGetImageRectInView**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the bounds of the image in the image display in view coordinates.
¬†
ImageReference **ImageDisplayGetInputColorTable**(ImageDisplay *imgDisp*)
Gets the input color table for the display.
¬†
ImageReference **ImageDisplayGetIntensityTransformation**(ImageDisplay *imgDisp*)
Gets the ITT of the display.
¬†
Number **ImageDisplayGetMinimumContrast**(ImageDisplay *imgDisp*)
Gets the minimum contrast of the display.
¬†
ImageReference **ImageDisplayGetOutputColorTable**(ImageDisplay *imgDisp*)

Gets the output color table for the display.
¬†
ROI **ImageDisplayGetROI**(ImageDisplay *imgDisp*, Number *index*)
Returns the given ROI on this image display.
¬†
Number **ImageDisplayGetROISelectionStyle**(ImageDisplay *imgDisp*, ROI *r*)
Gets the selection style of the roi in the imag display.
¬†
Number **ImageDisplayGetSurveyTechnique**(ImageDisplay *imgDisp*)
Gets the survey technique of the display.
¬†
Boolean **ImageDisplayIsCaptionOn**(ImageDisplay *imgDisp*)
Returns true if captions are drawn.
¬†
Boolean **ImageDisplayIsROISelected**(ImageDisplay *imgDisp*, ROI *roi*)
Determines whether the given ROI is selected on this image display.
¬†
Boolean **ImageDisplayIsValid**(ImageDisplay *imgDisp*)
Returns true if 'imageDisplay' points to a valid object.
¬†
ROI **ImageDisplayLookupROI**(ImageDisplay *imgDisp*, String *name*)
Returns the given ROI on this image display.
¬†
ROI **ImageDisplayLookupROIByID**(ImageDisplay *imgDisp*, Number *id*)
Returns the ROI with the given id on this image display.
¬†
void **ImageDisplayRemoveKeyListener**(ImageDisplay *imgDisp*, String *listener_key*)
Removes the listener script with the tag listener_key from the key listener list.
¬†
void **ImageDisplaySetCaptionOn**(ImageDisplay *imgDisp*, Boolean *on*)
Sets whether to draw captions.
¬†
void **ImageDisplaySetComplexMode**(ImageDisplay *imgDisp*, Number *mode*)
Sets the complex mode of the display.
¬†
void **ImageDisplaySetComplexModeRange**(ImageDisplay *imgDisp*, Number *range*)
Sets the parameter used in converting complex to real.
¬†
void **ImageDisplaySetContrastLimits**(ImageDisplay *imgDisp*, Number *low*, Number *hight*)
Sets the contrast limits of the display.
¬†
void **ImageDisplaySetContrastMode**(ImageDisplay *imgDisp*, Number *mode*)
Sets the contrast mode.
¬†
void **ImageDisplaySetContrastParameters**(ImageDisplay *imgDisp*, Number *bright*, Number *contrast*)
Gets the parameters for the contrast mode.
¬†
void **ImageDisplaySetDisplayedLayers**(ImageDisplay *imgDisp*, Number *start*, Number *end*)
Sets the layers that are summed into the display.
¬†
void **ImageDisplaySetDoAutoSurvey**(ImageDisplay *imgDisp*, Boolean *do_survey*)
Sets whether min and max are determined automatically.
¬†
void **ImageDisplaySetImageRect**(ImageDisplay *imgDisp*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the bounds of the image part of the image display.
¬†
void **ImageDisplaySetInputColorTable**(ImageDisplay *imgDisp*, ImageReference *clut*)
Sets the input color table of the display.
¬†
void **ImageDisplaySetIntensityTransformation**(ImageDisplay *imgDisp*, ImageReference *itt*)
Sets the ITT of the display.
¬†
void **ImageDisplaySetMinimumContrast**(ImageDisplay *imgDisp*, Number *contrast*)
Sets the minimum contrast of the display.

¬†

`void` **`ImageDisplaySetROISelected`**`( ImageDisplay` *`imgDisp`*`, ROI` *`roi`*`, Boolean` *`select`* `)`

Sets the selection status of the region of interest in the image display.

¬†

`void` **`ImageDisplaySetROISelectionStyle`**`( ImageDisplay` *`imgDisp`*`, ROI` *`r`*`, Number` *`style`* `)`

Sets the selection style of the roi in the imag display.

¬†

`void` **`ImageDisplaySetSurveyTechnique`**`( ImageDisplay` *`imgDisp`*`, Number` *`tech`* `)`

Sets the survey technique of the display.

¬†

Related Functions

`ImageDisplay` **`=`**`( ImageDisplay, Component )`

¬†

`ImageDisplay` **`=`**`( ImageDisplay, ! )`

¬†

`ImageDisplay` **`=`**`( ImageDisplay, ImageDisplay )`

¬†

`void` **`ApplyDataBar`**`( ImageDisplay` *`imgDisp`* `)`

Applies a data bar to the image.

¬†

`ImageDisplay` **`ImageDisplayNullify`**`( ! )`

¬†

`ImageDisplay` **`ImageDocumentAddImageDisplay`**`( ImageDocument` *`imgDoc`*`, ImageReference` *`image`*`, Number` *`displayType`* `)`

Adds the given image and an image display for it of the given type.

¬†

`ImageDisplay` **`ImageDocumentGetImageModeDisplay`**`( ImageDocument` *`imgDoc`* `)`

Gets the image display targeted by the current image mode.

¬†

`ImageReference` **`NewLiveFFT`**`( ImageDisplay` *`imageDisplay`*`, ROI` *`roi`*`, Boolean` *`reduce`* `)`

Creates a new live fft of the area in 'roi', that is reduced if 'reduce' is 'true'

¬†

`ImageReference` **`NewLiveHistogram`**`( ImageDisplay` *`imageDisplay`*`, ROI` *`roi`*`, Number` *`num_channels`* `)`

Creates a new live histogram of the area in 'roi', binned by 'num_channels'

¬†

`ImageReference` **`NewLiveProfile`**`( ImageDisplay` *`imageDisplay`*`, Number` *`start_x`*`, Number` *`start_y`*`, Number` *`end_x`*`, Number` *`end_y`*`, Number` *`width`* `)`

Creates a new live profile from (start_x,start_y) to (end_x,end_y)

¬†

Class RasterImageDisplay

Return to Top

Methods

`void` **`RasterImageDisplayAddThresholdToMask`**`( RasterImageDisplay` *`rid`*`, ImageReference` *`mask`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`

Sets the points in mask to 1 if they lie within the threshold.

¬†

`void` **`RasterImageDisplayGetThresholdLimits`**`( RasterImageDisplay` *`rid`*`, NumberVariable` *`low`*`, NumberVariable` *`high`* `)`

Gets the threshold limits of the display.

¬†

`Boolean` **`RasterImageDisplayIsThresholdOn`**`( RasterImageDisplay` *`rid`* `)`

Determines whether the thresholding overlay is on or off.

¬†

`void` **`RasterImageDisplaySetThresholdLimits`**`( RasterImageDisplay` *`rid`*`, Number` *`low`*`, Number` *`high`* `)`

Sets the threshold limits of the display.

¬†

`void` **`RasterImageDisplaySetThresholdOn`**`( RasterImageDisplay` *`rid`*`, Boolean` *`on`* `)`

Sets whether the thresholding overlay is on or off.

¬†

Related Functions

`RasterImageDisplay` **`=`**`( RasterImageDisplay` *`dst`*`, ! )`

¬†

`RasterImageDisplay` **`=`**`( RasterImageDisplay` *`dst`*`, RasterImageDisplay` *`rid`* `)`

¬†

`RasterImageDisplay` **`=`**`( RasterImageDisplay` *`dst`*`, ImageDisplay` *`id`* `)`

¬†

`ImageReference` **`CreateMaskFromAnnotations`**`( RasterImageDisplay` *`rid`*`, Number` *`filter_length`*`, Boolean` *`is_opaque`*`, NumberVariable` *`has_mask`* `)`

¬†

`RasterImageDisplay` **`RasterImageDisplayNullify`**`( ! )`

¬†

Class LinePlotImageDisplay

Return to Top

Methods

`Number` **`LinePlotImageDisplayCountSlices`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns the number of slices in the line plot.

¬†

`Number` **`LinePlotImageDisplayGetBaseIntensity`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns the base intensity of the line plot.

¬†

`void` **`LinePlotImageDisplayGetContrastLimits`**`( LinePlotImageDisplay` *`lpid`*`, NumberVariable` *`lowLimit`*`, NumberVariable` *`highLimit`* `)`

Gets the lowest and higest intensities displayed.

¬†

`void` **`LinePlotImageDisplayGetDisplayedChannels`**`( LinePlotImageDisplay` *`lpid`*`, NumberVariable` *`leftChannel`*`, NumberVariable` *`rightChannel`* `)`

Gets the leftmost and rightmost displayed channels.

¬†

`void` **`LinePlotImageDisplayGetDoAutoSurvey`**`( LinePlotImageDisplay` *`lpid`*`, NumberVariable` *`doAutoSurveyLow`*`, NumberVariable` *`doAutoSurveyHigh`* `)`

Gets whether to auto-survey is done on the high and low intensity limits.

¬†

`Number` **`LinePlotImageDisplayGetSlice`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns slice currently displayed at the bottom.

¬†

`void` **`LinePlotImageDisplayGetSliceComponentColor`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`slice_index`*`, Number` *`comp_index`*`, NumberVariable` *`r`*`, NumberVariable` *`g`*`, NumberVariable` *`b`* `)`

Returns the color of the 'comp_index'th component of the 'slice_index'th slice.

¬†

`Number` **`LinePlotImageDisplayGetSliceDrawingStyle`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`slice_index`* `)`

Returns the drawing style of the 'slice_index'th slice.

¬†

`void` **`LinePlotImageDisplayGetTrackingStyle`**`( LinePlotImageDisplay` *`lpid`*`, NumberVariable` *`track_style_x`*`, NumberVariable` *`track_style_y`* `)`

Gets the tracking style of the line plot.

¬†

`Boolean` **`LinePlotImageDisplayIsBackgroundOn`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns true if the background is erased.

¬†

`Boolean` **`LinePlotImageDisplayIsFilled`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns true if the line plot is filled.

¬†

`Boolean` **`LinePlotImageDisplayIsFrameOn`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns true if the frame is drawn.

¬†

`Boolean` **`LinePlotImageDisplayIsGridOn`**`( LinePlotImageDisplay` *`lpid`* `)`

Returns true if the grid is displayed on.

¬†

`void` **`LinePlotImageDisplaySetBackgroundOn`**`( LinePlotImageDisplay` *`lpid`*`, Boolean` *`on`* `)`

Sets whether to erase the background.

¬†

`void` **`LinePlotImageDisplaySetBaseIntensity`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`base_intensity`* `)`

Sets the base intensity of the line plot.

¬†

`void` **`LinePlotImageDisplaySetContrastLimits`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`lowLimit`*`, Number` *`highLimit`* `)`

Sets the lowest and highest intensities displayed.

¬†

`void` **`LinePlotImageDisplaySetDisplayedChannels`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`leftChannel`*`, Number` *`rightChannel`* `)`

Sets the leftmost and rightmost displayed channels.

¬†

`void` **`LinePlotImageDisplaySetDoAutoSurvey`**`( LinePlotImageDisplay` *`lpid`*`, Boolean` *`doAutoSurveyLow`*`, Boolean` *`doAutoSurveyHigh`* `)`

Sets whether to do auto-survey on the high and low intensity limits.

¬†

`void` **`LinePlotImageDisplaySetFilled`**`( LinePlotImageDisplay` *`lpid`*`, Boolean` *`on`* `)`

Sets whether to fill the lineplot.

¬†

`void` **`LinePlotImageDisplaySetFrameOn`**`( LinePlotImageDisplay` *`lpid`*`, Boolean` *`on`* `)`

Sets whether to draw the frame.

¬†

`void` **`LinePlotImageDisplaySetGridOn`**`( LinePlotImageDisplay` *`lpid`*`, Boolean` *`on`* `)`

Sets whether to draw the grid.

¬†

`void` **`LinePlotImageDisplaySetSlice`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`slice`* `)`

Sets the slice currently displayed at the bottom.

¬†

`void` **`LinePlotImageDisplaySetSliceComponentColor`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`slice_index`*`, Number` *`comp_index`*`, Number` *`r`*`, Number` *`g`*`, Number` *`b`* `)`

Sets the color of the 'comp_index'th component of the 'slice_index'th slice.

¬†

`void` **`LinePlotImageDisplaySetSliceDrawingStyle`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`slice_index`*`, Number` *`style`* `)`

Sets the drawing style of the 'slice_index'th slice.

¬†

`void` **`LinePlotImageDisplaySetTrackingStyle`**`( LinePlotImageDisplay` *`lpid`*`, Number` *`track_style_x`*`, Number` *`track_style_y`* `)`

Sets the tracking style of the line plot.

¬†

Related Functions

```
LinePlotImageDisplay =( LinePlotImageDisplay dst, LinePlotImageDisplay lpid )
      ¬†
LinePlotImageDisplay =( LinePlotImageDisplay dst, ! )
      ¬†
LinePlotImageDisplay =( LinePlotImageDisplay dst, ImageDisplay id )
      ¬†
LinePlotImageDisplay LinePlotImageDisplayNullify( ! )
      ¬†
```

Class SurfacePlotImageDisplay

Return to Top

Overview

The shape of a surface plot image display is given by the x,y offset of the x-axis, the x,y offset of the y-axis, and the y-offset of the z-axis, denoted '(x_axis_x,x_axis_y)', '(y_axis_x,y_axis_y)', and 'z_axis' in the script methods. In all of these offsets, positive x points to the left and positive y points down. Only the relative sizes of these values matter, for the coordinates are scaled to fit in the display bounds.

Methods

```
void SurfacePlotImageDisplayGetCubeAxes( SurfacePlotImageDisplay spid, NumberVariable x_axis_x,
NumberVariable x_axis_y, NumberVariable y_axis_x, NumberVariable y_axis_y, NumberVariable z_axis )
```
Gets the points describing the surface plot cube.
```
      ¬†
void SurfacePlotImageDisplayGetCubePoint( SurfacePlotImageDisplay spid, Number which_point,
NumberVariable x, NumberVariable y )
```
Gets the child coordinates of the cube point indicated by 'which_point'
```
      ¬†
Boolean SurfacePlotImageDisplayIsShadingOn( SurfacePlotImageDisplay spid )
```
Determines whether shading is on or off.
```
      ¬†
void SurfacePlotImageDisplaySetCubeAxes( SurfacePlotImageDisplay spid, Number x_axis_x, Number
x_axis_y, Number y_axis_x, Number y_axis_y, Number z_axis )
```
Sets the points describing the surface plot cube.
```
      ¬†
void SurfacePlotImageDisplaySetShadingOn( SurfacePlotImageDisplay spid, Boolean on )
```
Sets whether shading is on or off.
```
      ¬†
```

Related Functions

```
SurfacePlotImageDisplay =( SurfacePlotImageDisplay dst, ImageDisplay id )
      ¬†
SurfacePlotImageDisplay =( SurfacePlotImageDisplay dst, ! )
      ¬†
SurfacePlotImageDisplay =( SurfacePlotImageDisplay dst, SurfacePlotImageDisplay spid )
      ¬†
SurfacePlotImageDisplay SurfacePlotImageDisplayNullify( ! )
      ¬†
```

Class Window

Return to Top

Methods

`void` **`WindowClose`**`( DocumentWindow` *`window`*`, Boolean` *`verify`*`  )`
Closes the window, prompting the user if 'verify' is true.
¬†

`void` **`WindowGetContentBounds`**`( DocumentWindow` *`window`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`*`  )`
Gets the bounding rectangle of the content area of the 'window'.
¬†

`void` **`WindowGetContentPosition`**`( DocumentWindow` *`window`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*`  )`
Gets the position of the top-left corner of the content area of the 'window'.
¬†

`void` **`WindowGetContentSize`**`( DocumentWindow` *`window`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*`  )`
Gets the size of the content area of the 'window'.
¬†

`void` **`WindowGetFrameBounds`**`( DocumentWindow` *`window`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`*`  )`
Gets the bounding rectangle of the frame area of the 'window'.
¬†

`void` **`WindowGetFramePosition`**`( DocumentWindow` *`window`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*`  )`
Gets the position of the top-left corner of the frame area of the 'window'.
¬†

`void` **`WindowGetFrameSize`**`( DocumentWindow` *`window`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*`  )`
Gets the size of the frame area of the 'window'.
¬†

`void` **`WindowGetMousePosition`**`( DocumentWindow` *`window`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*`  )`
Gets the current position of the mouse in the windows coordinate system.
¬†

`String` **`WindowGetTitle`**`( DocumentWindow` *`window`*`  )`
Gets the title of the window.
¬†

`Number` **`WindowGetType`**`( DocumentWindow` *`window`*`  )`
Gets the type of thw window.
¬†

`void` **`WindowHide`**`( DocumentWindow` *`window`*`  )`
Hides the window.
¬†

`Boolean` **`WindowIsOpen`**`( DocumentWindow` *`window`*`  )`
Returns true if the window has not been closed.
¬†

`Boolean` **`WindowIsShown`**`( DocumentWindow` *`window`*`  )`
Returns true if the window is shown.
¬†

`Boolean` **`WindowIsValid`**`( DocumentWindow` *`window`*`  )`
Returns true if 'window' points to a valid object.
¬†

`void` **`WindowSelect`**`( DocumentWindow` *`window`*`  )`
Brings 'window' to the front.
¬†

`void` **`WindowSendBehind`**`( DocumentWindow` *`window`*`, DocumentWindow` *`behind_window`*`  )`
Sends 'window' behind 'behind_window'.
¬†

`void` **`WindowSetContentBounds`**`( DocumentWindow` *`window`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Sets the bounding rectangle of the content area of the 'window'.
¬†

`void` **`WindowSetContentPosition`**`( DocumentWindow` *`window`*`, Number` *`x`*`, Number` *`y`*` )`
Sets the position of the top-left corner of the content area of the 'window'.
¬†

`void` **`WindowSetContentSize`**`( DocumentWindow` *`window`*`, Number` *`x`*`, Number` *`y`*` )`
Sets the size of the content area of the 'window'.
¬†

`void` **`WindowSetFrameBounds`**`( DocumentWindow` *`window`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Sets the bounding rectangle of the frame area of the 'window'.
¬†

`void` **`WindowSetFramePosition`**`( DocumentWindow` *`window`*`, Number` *`x`*`, Number` *`y`*` )`
Sets the position of the top-left corner of the frame area of the 'window'.
¬†

`void` **`WindowSetFrameSize`**`( DocumentWindow` *`window`*`, Number` *`x`*`, Number` *`y`*` )`
Sets the size of the frame area of the 'window'.
¬†

`void` **`WindowSetTitle`**`( DocumentWindow` *`window`*`, String` *`title`*` )`
Sets the title of the window.
¬†

`void` **`WindowShow`**`( DocumentWindow` *`window`*` )`
Shows the window.
¬†

`void` **`WindowUpdate`**`( DocumentWindow` *`window`*` )`
Updates 'window's display.
¬†

Related Functions

`DocumentWindow` **`=`**`( DocumentWindow, ! )`
¬†

`DocumentWindow` **`=`**`( DocumentWindow, DocumentWindow )`
¬†

`void` **`EditorWindowAddText`**`( DocumentWindow` *`window`*`, String` *`text`*` )`
Appends the text to a editor window.
¬†

`void` **`EditorWindowGetFont`**`( DocumentWindow` *`window`*`, String` *`face_name`*`, NumberVariable` *`attributes`*`, NumberVariable` *`size`*`, NumberVariable` *`text_encoding`*` )`
Gets the font of a script window.
¬†

`String` **`EditorWindowGetText`**`( DocumentWindow` *`window`*` )`
Gets the text in an editor window.
¬†

`Boolean` **`EditorWindowPrint`**`( DocumentWindow` *`window`*` )`
Prints the editor window.
¬†

`void` **`EditorWindowSaveToFile`**`( DocumentWindow` *`window`*`, String` *`path`*` )`
Saves the editor window to the specified path.
¬†

`void` **`EditorWindowSetFont`**`( DocumentWindow` *`window`*`, String` *`face_name`*`, Number` *`attributes`*`, Number` *`size`*`, Number` *`text_encoding`*` )`
Sets the font of a script window.
¬†

`void` **`EditorWindowSetText`**`( DocumentWindow` *`window`*`, String` *`text`*` )`
Sets the text in an editor window.
¬†

`DocumentWindow` **`GetApplicationWindow`**`( )`
Gets the application window.
¬†

`DocumentWindow` **`GetDocumentWindow`**`( Number` *`index`*` )`
Gets the 'index'th document window.

¬†
`DocumentWindow` **`GetDocumentWindowByTitle`**`( String` *`name`* `)`
Gets the document window named 'name'.
¬†
`DocumentWindow` **`GetFloatingWindow`**`( Number` *`index`* `)`
Gets the 'index'th floating window.
¬†
`DocumentWindow` **`GetNthDocumentWindowOfType`**`( Number` *`type`*`, Number` *`index`* `)`
Returns the 'index'th document window of type 'type'.
¬†
`DocumentWindow` **`GetResultsWindow`**`( Boolean` *`open`* `)`
Gets the results window. If the window is not open, and 'open' is true, the results window is opened.
¬†
`ImageDocument` **`ImageWindowGetImageDocument`**`( DocumentWindow` *`window`* `)`
Gets the image document displayed in the window.
¬†
`DocumentWindow` **`NewScriptWindow`**`( String` *`title`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Creates a new editor window.
¬†
`DocumentWindow` **`NewScriptWindowFromFile`**`( String` *`file_name`*`, String` *`font_name`*`, Number` *`attributes`*`, Number` *`size`*`, Number` *`encoding`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Opens a file into a script window.
¬†
`DocumentWindow` **`NewScriptWindowFromFile`**`( String` *`file_name`* `)`
Opens a file into a script window.
¬†
`DocumentWindow` **`NewScriptWindowFromFile`**`( String` *`file_name`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Opens a file into a script window.
¬†
`DocumentWindow` **`NewScriptWindowFromFile`**`( String` *`file_name`*`, String` *`font_name`*`, Number` *`attributes`*`, Number` *`size`*`, Number` *`encoding`* `)`
Opens a file into a script window.
¬†
`void` **`ScriptWindowExecute`**`( DocumentWindow` *`window`* `)`
Executes the script in the script window.
¬†
`DocumentWindow` **`WindowNullify`**`( ! )`
¬†

Class TagGroup

Return to Top

Methods

`TagGroup` **`TagGroupAddLabeledTagGroup`**`( TagGroup` *`tagGroup`*`, String` *`label`*`, TagGroup` *`newGroup`* `)`
Adds 'newGroup' to 'tagGroup' at the label 'label'.
¬†
`TagGroup` **`TagGroupAddTagGroupAfter`**`( TagGroup` *`tagList`*`, Number` *`ref_index`*`, TagGroup` *`newGroup`* `)`
Adds 'newGroup' to 'tagList' after index 'ref_index'.
¬†
`TagGroup` **`TagGroupAddTagGroupAtBeginning`**`( TagGroup` *`tagList`*`, TagGroup` *`newGroup`* `)`
Adds 'newGroup' to the beginning of 'tagList'.
¬†

TagGroup **TagGroupAddTagGroupAtEnd**(TagGroup *tagList*, TagGroup *newGroup*)
Adds 'newGroup' to the end of 'tagList'.
¬†
TagGroup **TagGroupAddTagGroupBefore**(TagGroup *tagList*, Number *ref_index*, TagGroup *newGroup*)
Adds 'newGroup' to 'tagList' before index 'ref_index'.
¬†
TagGroup **TagGroupClone**(TagGroup *tagGroup*)
Returns an identical copy of 'tagGroup' and its sub-tags.
¬†
Number **TagGroupCopyTag**(TagGroup *tagGroup*, TagGroup *srcGroup*, Number *srcIndex*)
Copies the 'srcIndex'th tag in 'srcGroup' to 'tagGroup'.
¬†
void **TagGroupCopyTagsFrom**(TagGroup *tagGroup*, TagGroup *srcGroup*)
Copies tags in 'srcGroup' to 'tagGroup'.
¬†
void **TagGroupCopyTagToIndex**(TagGroup *tagGroup*, Number *dstIndex*, TagGroup *srcGroup*, Number *srcIndex*)
Copies data in the 'srcIndex'th tag in 'srcGroup' to the 'dstIndex'th tag in 'tagGroup'.
¬†
Number **TagGroupCountTags**(TagGroup *tagGroup*)
Returns the number of sub-tags in this tag group.
¬†
TagGroup **TagGroupCreateGroupTagAfter**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group after 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateGroupTagAtBeginning**(TagGroup *tagList*)
Creates a new tag group at the beginning of 'tagList'.
¬†
TagGroup **TagGroupCreateGroupTagAtEnd**(TagGroup *tagList*)
Creates a new tag group at the end of 'tagList'.
¬†
TagGroup **TagGroupCreateGroupTagBefore**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group before 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateListTagAfter**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group after 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateListTagAtBeginning**(TagGroup *tagList*)
Creates a new tag group at the beginning of 'tagList'.
¬†
TagGroup **TagGroupCreateListTagAtEnd**(TagGroup *tagList*)
Creates a new tag group at the end of 'tagList'.
¬†
TagGroup **TagGroupCreateListTagBefore**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group before 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateNewLabeledGroup**(TagGroup *tagGroup*, String *label*)
Adds a new tag group at label 'label' and returns the new group.
¬†
TagGroup **TagGroupCreateNewLabeledList**(TagGroup *tagGroup*, String *label*)
Adds a new tag list at label 'label' and returns the new group.
¬†
Number **TagGroupCreateNewLabeledTag**(TagGroup *tagGroup*, String *label*)
Creates a new labeled tag and returns its index.
¬†
Number **TagGroupCreateNewTagAfter**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag after 'ref_index' in 'tagList'.
¬†
Number **TagGroupCreateNewTagAtBeginning**(TagGroup *tagList*)
Creates a new tag at the beginning of 'tagList'.
¬†
Number **TagGroupCreateNewTagAtEnd**(TagGroup *tagList*)
Creates a new tag at the end of 'tagList'.
¬†
Number **TagGroupCreateNewTagBefore**(TagGroup *tagList*, Number *ref_index*)

Creates a new tag before 'ref_index' in 'tagList'.
¬†
void **TagGroupDeleteAllTags**(TagGroup *tagGroup*)
Deletes all the tags in 'tagGroup'.
¬†
void **TagGroupDeleteTagWithIndex**(TagGroup *taGroup*, Number *index*)
Deletes the tag at index 'index'.
¬†
void **TagGroupDeleteTagWithLabel**(TagGroup *tagGroup*, String *tagPath*)
Deletes the tag labelled by the path 'tagPath'.
¬†
Boolean **TagGroupDoesTagExist**(TagGroup *tagGroup*, String *tagPath*)
Finds the tag group and index corresponding to the tag referenced by 'tagPath' in 'tagGroup'.
¬†
void **TagGroupExecuteScriptGroup**(TagGroup *tagGroup*, String *form*)
Execute a group of script functions in 'tagGroup'. The actual scripts executed will be formed by sprintf'ing into the 'form' parameter. The form parameter should contain exactly one '%s' into which the function name will be inserted.
¬†
Boolean **TagGroupGetIndexedTagAsArray**(TagGroup *tagGroup*, Number *index*, ImageReference *image*)
Gets the data at 'index' in 'tagGroup' as an array of data in 'image'.
¬†
Boolean **TagGroupGetIndexedTagAsBoolean**(TagGroup *tagGroup*, Number *index*, NumberVariable *val*)
Gets the data at 'index' in 'tagGroup' as a boolean.
¬†
Boolean **TagGroupGetIndexedTagAsDouble**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a double.
¬†
Boolean **TagGroupGetIndexedTagAsDoubleComplex**(TagGroup *tagGroup*, Number *index*, ComplexNumberVariable *c*)
Gets the data at 'index' in 'tagGroup' as a double complex.
¬†
Boolean **TagGroupGetIndexedTagAsEightBitColor**(TagGroup *tagGroup*, Number *index*, RGBNumberVariable *c*)
Gets the data at 'index' in 'tagGroup' as an eight bit color.
¬†
Boolean **TagGroupGetIndexedTagAsFloat**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a float.
¬†
Boolean **TagGroupGetIndexedTagAsFloatComplex**(TagGroup *tagGroup*, Number *index*, ComplexNumberVariable *c*)
Gets the data at 'index' in 'tagGroup' as a float complex.
¬†
Boolean **TagGroupGetIndexedTagAsFloatPoint**(TagGroup *tagGroup*, Number *index*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'index' in 'tagGroup' as a float point.
¬†
Boolean **TagGroupGetIndexedTagAsFloatRect**(TagGroup *tagGroup*, Number *index*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'index' in 'tagGroup' as a float rect.
¬†
Boolean **TagGroupGetIndexedTagAsLong**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a long.
¬†
Boolean **TagGroupGetIndexedTagAsLongPoint**(TagGroup *tagGroup*, Number *index*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'index' in 'tagGroup' as a long point.
¬†
Boolean **TagGroupGetIndexedTagAsLongRect**(TagGroup *tagGroup*, Number *index*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'index' in 'tagGroup' as a long rect.
¬†
Boolean **TagGroupGetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a real number.
¬†
Boolean **TagGroupGetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, ComplexNumberVariable *number*)

Gets the data at 'index' in 'tagGroup' as a complex number.
¬†
Boolean **TagGroupGetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, RGBNumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a RGB number.
¬†
Boolean **TagGroupGetIndexedTagAsShort**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a short.
¬†
Boolean **TagGroupGetIndexedTagAsShortPoint**(TagGroup *tagGroup*, Number *index*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'index' in 'tagGroup' as a short point.
¬†
Boolean **TagGroupGetIndexedTagAsShortRect**(TagGroup *tagGroup*, Number *index*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'index' in 'tagGroup' as a short rect.
¬†
Boolean **TagGroupGetIndexedTagAsString**(TagGroup *tagGroup*, Number *index*, String *str*)
Gets the data at 'index' in 'tagGroup' as a string.
¬†
Boolean **TagGroupGetIndexedTagAsTagGroup**(TagGroup *tagGroup*, Number *index*, TagGroup *subGroup*)
Gets the data at 'index' in 'TagGroup' as a group.
¬†
Boolean **TagGroupGetIndexedTagAsText**(TagGroup *tagGroup*, Number *index*, String *str*)
Gets the data at 'index' in 'tagGroup' as a string.
¬†
Boolean **TagGroupGetIndexedTagAsUInt16**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a 16-bit unsigned integer.
¬†
Boolean **TagGroupGetIndexedTagAsUInt32**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a 32-bit unsigned integer.
¬†
TagGroup **TagGroupGetOrCreateTagGroup**(TagGroup *tagGroup*, String *tagPath*)
Gets the tag group named by 'tagPath', or creates a new such group and all necessary intermediate groups.
¬†
TagGroup **TagGroupGetOrCreateTagList**(TagGroup *tagGroup*, String *tagPath*)
Gets the tag list named by 'tagPath', or creates a new such list and all necessary intermediate groups.
¬†
Number **TagGroupGetSeeds**(TagGroup *tagGroup*)
Gets a set of seeds that describe the tag group.
¬†
Boolean **TagGroupGetTagAsArray**(TagGroup *tagGroup*, String *tagPath*, ImageReference *image*)
Gets the data at 'tagPath' in 'tagGroup' as an array of data in 'image'.
¬†
Boolean **TagGroupGetTagAsBoolean**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *val*)
Gets the data at 'tagPath' in 'tagGroup' as a boolean.
¬†
Boolean **TagGroupGetTagAsDouble**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a double.
¬†
Boolean **TagGroupGetTagAsDoubleComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumberVariable *c*)
Gets the data at 'tagPath' in 'tagGroup' as a double complex.
¬†
Boolean **TagGroupGetTagAsEightBitColor**(TagGroup *tagGroup*, String *tagPath*, RGBNumberVariable *c*)
Gets the data at 'tagPath' in 'tagGroup' as an eight bit color.
¬†
Boolean **TagGroupGetTagAsFloat**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a float.
¬†
Boolean **TagGroupGetTagAsFloatComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumberVariable *c*)
Gets the data at 'tagPath' in 'tagGroup' as a float complex.
¬†
Boolean **TagGroupGetTagAsFloatPoint**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'tagPath' in 'tagGroup' as a short point.
¬†

Boolean **TagGroupGetTagAsFloatRect**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'tagPath' in 'tagGroup' as a short rect.
¬†

Boolean **TagGroupGetTagAsLong**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a long.
¬†

Boolean **TagGroupGetTagAsLongPoint**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'tagPath' in 'tagGroup' as a long point.
¬†

Boolean **TagGroupGetTagAsLongRect**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'tagPath' in 'tagGroup' as a long rect.
¬†

Boolean **TagGroupGetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, ComplexNumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a complex number.
¬†

Boolean **TagGroupGetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a real number.
¬†

Boolean **TagGroupGetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, RGBNumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a rgb number.
¬†

Boolean **TagGroupGetTagAsRGBUInt16**(TagGroup *tagGroup*, Number *index*, NumberVariable *r*, NumberVariable *g*, NumberVariable *b*)
Gets the data at 'index' in 'tagGroup' as a 16-bit rgb value.
¬†

Boolean **TagGroupGetTagAsRGBUInt16**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *r*, NumberVariable *g*, NumberVariable *b*)
Gets the data at 'tagPath' in 'tagGroup' as a 16-bit rgb value.
¬†

Boolean **TagGroupGetTagAsShort**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a short.
¬†

Boolean **TagGroupGetTagAsShortPoint**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'tagPath' in 'tagGroup' as a short point.
¬†

Boolean **TagGroupGetTagAsShortRect**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'tagPath' in 'tagGroup' as a short rect.
¬†

Boolean **TagGroupGetTagAsString**(TagGroup *tagGroup*, String *tagPath*, String *str*)
Gets the data at 'tagPath' in 'tagGroup' as a string.
¬†

Boolean **TagGroupGetTagAsTagGroup**(TagGroup *tagGroup*, String *tagPath*, TagGroup *subGroup*)
Gets the data at 'tagPath' in 'TagGroup' as a group.
¬†

Boolean **TagGroupGetTagAsText**(TagGroup *tagGroup*, String *tagPath*, String *str*)
Gets the data at 'tagPath' in 'tagGroup' as a string.
¬†

Boolean **TagGroupGetTagAsUInt16**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a 16-bit unsigned integer.
¬†

Boolean **TagGroupGetTagAsUInt32**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a 32-bit unsigned integer.
¬†

String **TagGroupGetTagLabel**(TagGroup *tagGroup*, Number *index*)
Gets the label of the 'index'th tag in the tag group.
¬†

Number **TagGroupGetTagSize**(TagGroup *tagGroup*, Number *index*)
Gets the size of the tag.
¬†

Number **TagGroupGetTagType**(TagGroup *tagGroup*, Number *index*, Number *type_index*)
Returns the 'type_index'th element of the tag's type.

¬†

Number **TagGroupGetTagTypeLength**(TagGroup *tagGroup*, Number *index*)

Returns number of elements in the tag's type.

¬†

Boolean **TagGroupHasChangedSince**(TagGroup *tagGroup*, Number *seeds*)

Returns true if the tag group has changed since 'seeds' was constructed.

¬†

void **TagGroupInsertTagAsArray**(TagGroup *tagGroup*, Number *ref_index*, ImageReference *image*)

Inserts new data before 'ref_index' in 'tagGroup' as an array of data in 'image'.

¬†

void **TagGroupInsertTagAsBoolean**(TagGroup *tagGroup*, Number *ref_index*, Boolean *val*)

Inserts new data before 'ref_index' in 'tagGroup' as a boolean.

¬†

void **TagGroupInsertTagAsDouble**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a double.

¬†

void **TagGroupInsertTagAsDoubleComplex**(TagGroup *tagGroup*, Number *ref_index*, ComplexNumber *c*)

Inserts new data before 'ref_index' in 'tagGroup' as a double complex.

¬†

void **TagGroupInsertTagAsEightBitColor**(TagGroup *tagGroup*, Number *ref_index*, RGBNumber *c*)

Inserts new data before 'ref_index' in 'tagGroup' as an eight bit color.

¬†

void **TagGroupInsertTagAsFloat**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a float.

¬†

void **TagGroupInsertTagAsFloatComplex**(TagGroup *tagGroup*, Number *ref_index*, ComplexNumber *c*)

Inserts new data before 'ref_index' in 'tagGroup' as a float complex.

¬†

void **TagGroupInsertTagAsFloatPoint**(TagGroup *tagGroup*, Number *ref_index*, Number *x*, Number *y*)

Inserts new data before 'ref_index' in 'tagGroup' as a float point.

¬†

void **TagGroupInsertTagAsFloatRect**(TagGroup *tagGroup*, Number *ref_index*, Number *t*, Number *l*, Number *b*, Number *r*)

Inserts new data before 'ref_index' in 'tagGroup' as a float rect.

¬†

void **TagGroupInsertTagAsLong**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a long.

¬†

void **TagGroupInsertTagAsLongPoint**(TagGroup *tagGroup*, Number *ref_index*, Number *x*, Number *y*)

Inserts new data before 'ref_index' in 'tagGroup' as a long point.

¬†

void **TagGroupInsertTagAsLongRect**(TagGroup *tagGroup*, Number *ref_index*, Number *t*, Number *l*, Number *b*, Number *r*)

Inserts new data before 'ref_index' in 'tagGroup' as a long rect.

¬†

void **TagGroupInsertTagAsNumber**(TagGroup *tagGroup*, Number *ref_index*, RGBNumber *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a RGB number.

¬†

void **TagGroupInsertTagAsNumber**(TagGroup *tagGroup*, Number *ref_index*, ComplexNumber *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a complex number.

¬†

void **TagGroupInsertTagAsNumber**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a real number.

¬†

void **TagGroupInsertTagAsRGBUInt16**(TagGroup *tagGroup*, Number *ref_index*, Number *r*, Number *g*, Number *b*)

Inserts new data before 'ref_index' in 'tagGroup' as a 16-bit rgb value.

¬†

void **TagGroupInsertTagAsShort**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)

Inserts new data before 'ref_index' in 'tagGroup' as a short.

¬†

void **TagGroupInsertTagAsShortPoint**(TagGroup *tagGroup*, Number *ref_index*, Number *x*, Number *y*)

Inserts new data before 'ref_index' in 'tagGroup' as a short point.

¬†

void **TagGroupInsertTagAsShortRect**(TagGroup *tagGroup*, Number *ref_index*, Number *t*, Number *l*, Number

b, Number *r*)
Inserts new data before 'ref_index' in 'tagGroup' as a short rect.
¬†
void **TagGroupInsertTagAsString**(TagGroup *tagGroup*, Number *ref_index*, String *s*)
Inserts new data before 'ref_index' in 'tagGroup' as a string.
¬†
void **TagGroupInsertTagAsTagGroup**(TagGroup *tagGroup*, Number *ref_index*, TagGroup *subGroup*)
Inserts new data before 'ref_index' in 'TagGroup' as a group.
¬†
void **TagGroupInsertTagAsText**(TagGroup *tagGroup*, Number *ref_index*, String *s*)
Inserts new data before 'ref_index' in 'tagGroup' as a string.
¬†
void **TagGroupInsertTagAsUInt16**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a 16-bit unsigned integer.
¬†
void **TagGroupInsertTagAsUInt32**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a 32-bit unsigned integer.
¬†
Boolean **TagGroupIsList**(TagGroup *tagGroup*)
Returns true if the tag group is a list.
¬†
Boolean **TagGroupIsOpen**(TagGroup *tagGroup*)
Returns whether 'tagGroup' is open or not.
¬†
Boolean **TagGroupIsValid**(TagGroup *tagGroup*)
Returns true if 'tagGroup' references a valid object.
¬†
Boolean **TagGroupLoadFromFile**(TagGroup *tagGroup*, String *path*)
Loads the contents of the file specified by 'path' into the tag group.
¬†
Boolean **TagGroupLoadFromFileWithLabel**(TagGroup *tagGroup*, String *path*, String *label*)
Loads the contents of the file specified by 'path' into the tag group and returns the label, if any.
¬†
Boolean **TagGroupLoadFromStream**(TagGroup *tagGroup*, ScriptObject *stream*)
Loads the contents of the stream specified by 'stream' into the tag group.
¬†
Boolean **TagGroupLoadFromStreamWithLabel**(TagGroup *tagGroup*, ScriptObject *stream*, String *label*)
Loads the contents of the stream specified by 'stream' into the tag group and returns the label, if any.
¬†
void **TagGroupMarkAsChanged**(TagGroup *tagGroup*)
Marks 'tagGroup' as having beein modified.
¬†
DocumentWindow **TagGroupOpenBrowserWindow**(TagGroup *tagGroup*, Boolean *isFileBased*)
Opens a browser window for the tag group.
¬†
Number **TagGroupParseAndCreateTagPath**(TagGroup *tagGroup*, String *tagPath*, TagGroup *parentGroup*, String *label*)
Finds the tag group and index corresponding to the tag referenced by 'tagPath' in 'tagGroup'.
¬†
Number **TagGroupParseTagPath**(TagGroup *tagGroup*, String *tagPath*, TagGroup *parentGroup*, String *label*)
Finds the tag group and index corresponding to the tag referenced by 'tagPath' in 'tagGroup'.
¬†
void **TagGroupReadIndexedTagDataFromStream**(TagGroup *tagGroup*, Number *index*, ScriptObject *stream*, Number *stream_endianness*)
Reads data from the stream into the indicated tag. The tag type determines how the data is read, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
void **TagGroupReadTagDataFromStream**(TagGroup *tagGroup*, String *tag_path*, ScriptObject *stream*, Number *stream_endianness*)
Reads data from the stream into the indicated tag. The tag type determines how the data is read, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
void **TagGroupReleaseSeeds**(TagGroup *tagGroup*, Number *seeds*)
Releases the seeds returned by 'TagGroupGetSeeds'.

¬†
void **TagGroupReplaceTagsWithCopy**(TagGroup *tagGroup*, TagGroup *srcGroup*)
Deletes all tags in 'tagGroup' and copies tags in 'srcGroup' to 'tagGroup'.
¬†
void **TagGroupSaveToFile**(TagGroup *tagGroup*, String *path*)
Saves the contents of the tag group to the file specified by 'path'.
¬†
void **TagGroupSaveToFileWithLabel**(TagGroup *tagGroup*, String *path*, String *label*)
Saves the contents of the tag group and the label 'label' to the file specified by 'path'.
¬†
void **TagGroupSaveToStream**(TagGroup *tagGroup*, ScriptObject *stream*)
Saves the contents of the tag group to the stream specified by 'stream'.
¬†
void **TagGroupSaveToStreamWithLabel**(TagGroup *tagGroup*, ScriptObject *stream*, String *label*)
Saves the contents of the tag group and the label 'label' to the stream specified by 'stream'.
¬†
void **TagGroupSetIndexedTagAsArray**(TagGroup *tagGroup*, Number *index*, ImageReference *image*)
Set the data at 'index' in 'tagGroup' as an array of data in 'image'.
¬†
void **TagGroupSetIndexedTagAsBoolean**(TagGroup *tagGroup*, Number *index*, Boolean *val*)
Sets the data at 'index' in 'tagGroup' as a boolean.
¬†
void **TagGroupSetIndexedTagAsDouble**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a double.
¬†
void **TagGroupSetIndexedTagAsDoubleComplex**(TagGroup *tagGroup*, Number *index*, ComplexNumber *c*)
Sets the data at 'index' in 'tagGroup' as a double complex.
¬†
void **TagGroupSetIndexedTagAsEightBitColor**(TagGroup *tagGroup*, Number *index*, RGBNumber *c*)
Sets the data at 'index' in 'tagGroup' as an eight bit color.
¬†
void **TagGroupSetIndexedTagAsFloat**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a float.
¬†
void **TagGroupSetIndexedTagAsFloatComplex**(TagGroup *tagGroup*, Number *index*, ComplexNumber *c*)
Sets the data at 'index' in 'tagGroup' as a float complex.
¬†
void **TagGroupSetIndexedTagAsFloatPoint**(TagGroup *tagGroup*, Number *index*, Number *x*, Number *y*)
Sets the data at 'index' in 'tagGroup' as a float point.
¬†
void **TagGroupSetIndexedTagAsFloatRect**(TagGroup *tagGroup*, Number *index*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'index' in 'tagGroup' as a float rect.
¬†
void **TagGroupSetIndexedTagAsLong**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a long.
¬†
void **TagGroupSetIndexedTagAsLongPoint**(TagGroup *tagGroup*, Number *index*, Number *x*, Number *y*)
Sets the data at 'index' in 'tagGroup' as a long point.
¬†
void **TagGroupSetIndexedTagAsLongRect**(TagGroup *tagGroup*, Number *index*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'index' in 'tagGroup' as a long rect.
¬†
void **TagGroupSetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, RGBNumber *number*)
Sets the data at 'index' in 'tagGroup' as a RGB number.
¬†
void **TagGroupSetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, ComplexNumber *number*)
Sets the data at 'index' in 'tagGroup' as a complex number.
¬†
void **TagGroupSetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a real number.
¬†
void **TagGroupSetIndexedTagAsRGBUInt16**(TagGroup *tagGroup*, Number *index*, Number *r*, Number *g*, Number *b*)

Sets the data at 'index' in 'tagGroup' as a 16-bit rgb value.
¬†
void **TagGroupSetIndexedTagAsShort**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a short.
¬†
void **TagGroupSetIndexedTagAsShortPoint**(TagGroup *tagGroup*, Number *index*, Number *x*, Number *y*)
Sets the data at 'index' in 'tagGroup' as a short point.
¬†
void **TagGroupSetIndexedTagAsShortRect**(TagGroup *tagGroup*, Number *index*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'index' in 'tagGroup' as a short rect.
¬†
void **TagGroupSetIndexedTagAsString**(TagGroup *tagGroup*, Number *index*, String *s*)
Sets the data at 'index' in 'tagGroup' as a string.
¬†
void **TagGroupSetIndexedTagAsTagGroup**(TagGroup *tagGroup*, Number *index*, TagGroup *subGroup*)
Sets the data at 'index' in 'TagGroup' as a group.
¬†
void **TagGroupSetIndexedTagAsText**(TagGroup *tagGroup*, Number *index*, String *s*)
Sets the data at 'index' in 'tagGroup' as a string.
¬†
void **TagGroupSetIndexedTagAsUInt16**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a 16-bit unsigned integer.
¬†
void **TagGroupSetIndexedTagAsUInt32**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a 32-bit unsigned integer.
¬†
void **TagGroupSetIsOpen**(TagGroup *tagGroup*, Boolean *is_open*)
Sets whether 'tagGroup' is open or not.
¬†
void **TagGroupSetTagAsArray**(TagGroup *tagGroup*, String *tagPath*, ImageReference *image*)
Set the data at 'tagPath' in 'tagGroup' as an array of data in 'image'.
¬†
void **TagGroupSetTagAsBoolean**(TagGroup *tagGroup*, String *tagPath*, Boolean *val*)
Sets the data at 'tagPath' in 'tagGroup' as a boolean.
¬†
void **TagGroupSetTagAsDouble**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a double.
¬†
void **TagGroupSetTagAsDoubleComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumber *c*)
Sets the data at 'tagPath' in 'tagGroup' as a double complex.
¬†
void **TagGroupSetTagAsEightBitColor**(TagGroup *tagGroup*, String *tagPath*, RGBNumber *c*)
Sets the data at 'tagPath' in 'tagGroup' as an eight bit color.
¬†
void **TagGroupSetTagAsFloat**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a float.
¬†
void **TagGroupSetTagAsFloatComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumber *c*)
Sets the data at 'tagPath' in 'tagGroup' as a float complex.
¬†
void **TagGroupSetTagAsFloatPoint**(TagGroup *tagGroup*, String *tagPath*, Number *x*, Number *y*)
Sets the data at 'tagPath' in 'tagGroup' as a float point.
¬†
void **TagGroupSetTagAsFloatRect**(TagGroup *tagGroup*, String *tagPath*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'tagPath' in 'tagGroup' as a float rect.
¬†
void **TagGroupSetTagAsLong**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a long.
¬†
void **TagGroupSetTagAsLongPoint**(TagGroup *tagGroup*, String *tagPath*, Number *x*, Number *y*)
Sets the data at 'tagPath' in 'tagGroup' as a long point.
¬†
void **TagGroupSetTagAsLongRect**(TagGroup *tagGroup*, String *tagPath*, Number *t*, Number *l*, Number *b*,

Number *r*)
Sets the data at 'tagPath' in 'tagGroup' as a long rect.
¬†

void **TagGroupSetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, RGBNumber *number*)
Sets the data at 'tagPath' in 'tagGroup' as a RGB number.
¬†

void **TagGroupSetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, ComplexNumber *number*)
Sets the data at 'tagPath' in 'tagGroup' as a complex number.
¬†

void **TagGroupSetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a real number.
¬†

void **TagGroupSetTagAsRGBUInt16**(TagGroup *tagGroup*, String *tagPath*, Number *r*, Number *g*, Number *b*)
Sets the data at 'tagPath' in 'tagGroup' as a 16-bit rgb value.
¬†

void **TagGroupSetTagAsShort**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a short.
¬†

void **TagGroupSetTagAsShortPoint**(TagGroup *tagGroup*, String *tagPath*, Number *x*, Number *y*)
Sets the data at 'tagPath' in 'tagGroup' as a short point.
¬†

void **TagGroupSetTagAsShortRect**(TagGroup *tagGroup*, String *tagPath*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'tagPath' in 'tagGroup' as a short rect.
¬†

void **TagGroupSetTagAsString**(TagGroup *tagGroup*, String *tagPath*, String *s*)
Sets the data at 'tagPath' in 'tagGroup' as a string.
¬†

void **TagGroupSetTagAsTagGroup**(TagGroup *tagGroup*, String *tagPath*, TagGroup *subGroup*)
Sets the data at 'tagPath' in 'TagGroup' as a group.
¬†

void **TagGroupSetTagAsText**(TagGroup *tagGroup*, String *tagPath*, String *s*)
Sets the data at 'tagPath' in 'tagGroup' as a string.
¬†

void **TagGroupSetTagAsUInt16**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a 16-bit unsigned integer.
¬†

void **TagGroupSetTagAsUInt32**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a 32-bit unsigned integer.
¬†

void **TagGroupSetTagRGBBitmap**(TagGroup *tagGroup*, String *tagPath*, ImageReference *image*)
Sets the data at 'tagPath' in 'tagGroup' as a RGB bitmap.
¬†

void **TagGroupWriteIndexedTagDataToStream**(TagGroup *tagGroup*, Number *index*, ScriptObject *stream*, Number *stream_endianness*)
Writes data from the indicated tag into the stream. The tag type determines how the data is written, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†

void **TagGroupWriteTagDataToStream**(TagGroup *tagGroup*, String *tag_path*, ScriptObject *stream*, Number *stream_endianness*)
Reads data from the indicated tag into the stream. The tag type determines how the data is written, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†

Related Functions

TagGroup **=**(TagGroup *tagGroup1*, !)
¬†

TagGroup **=**(TagGroup *tagGroup1*, TagGroup *tagGroup2*)
¬†

void **AddTagsToPackage**(TagGroup *tags*, String *packageName*, Number *packageLevel*, String *identifier*)
Install the tags into the package. The identifier is used to identify the tags in the packages. Clients should take care to use unique identifiers. See the Java model of naming classes.
¬†

`void` **`ClipboardSetAsTagGroup`**`( TagGroup` *`tagGroup`* `)`
Sets the contents of the clipboard to the tag group.
¬†
`TagGroup` **`GetFilesInDirectory`**`( String` *`path`*`, Number` *`search_flags`* `)`
Returns a tag group containing a list of the file names in the directory 'dir_path'
¬†
`TagGroup` **`GetPackageTags`**`( String` *`identifier`* `)`
Return the tags specified by identifier. The identifier is used to identify tags loaded with a specific package.
¬†
`TagGroup` **`GetPersistentTagGroup`**`( )`
Gets the persistent tag group.
¬†
`TagGroup` **`NewTagGroup`**`( )`
Creates an empty tag group.
¬†
`TagGroup` **`NewTagList`**`( )`
Creates an empty tag list.
¬†
`TagGroup` **`TagGroupNullify`**`( ! )`
¬†
`TagGroup` **`Test_ReferenceCount_V1`**`( ROI` *`roi_out`* `)`
¬†
`TagGroup` **`Test_ReferenceCount_V2`**`( ROI` *`roi_out`* `)`
¬†

Class ImageDocument

Return to Top

Overview

Image documents are the base containers for image displays and other components. Each image document has a root component that can be accessed via **`ImageDocumentGetRootComponent`**, in which image displays may be placed. The child coordinate system of this annotation is called **page** coordinates. This coordinate system has arbitrary units and origin, but these are given a physical interpretation when the document is first viewed in page mode or the method **`ImageDocumentEnsurePlacedOnPage`** is first called, at which point **`ImageDocumentGetPageBounds`** will return the rectangle of the printable page expressed in page coordinates.

An image document may be viewed in its entirety in page mode, or one of its top-level image displays may be viewed in image mode. The coordinate system used to express the the portion of the document currently viewed in the window is **view** coordinates. In page mode this is equivalent to **page** coordinates; in image mode it has no equivalent, but methods like **`ComponentGetBoundingRectInView`** can be used to interpret the coordinates in terms of the visible components. View coordinates can be related to window coordinates by finding the currently visible area in view coordinates using **`ImageDocumentGetVisibleViewRect`**, and then by finding the visible area in screen coordinates using **`ImageDocumentGetWindow`** to get the document's window and **`DocumentWindowGetSize`** to get the size of the interior of the window. Each image document mode has a reference point size, which is the height of one text point, and a minimum resolution, which is the size of a pixel in the appropriate drawing surface. The reference point size in view coordinates is given by **`ImageDocumentGetReferencePointSize`**, and the minimum point size in view coordinates by **`ImageDocumentGetMinimumPointSize`**. In image mode, both the reference point and the minimum point are one screen pixel. In page mode, the reference point size is the size of a dot at 72dpi (before any scaling specified in the print dialog), and the minimum point size is the dot size of the selected printer.

Methods

`void` **`ImageDocumentAddImage`**`( ImageDocument` *`imgDoc`*`, ImageReference` *`image`* `)`
Adds the given image to the list maintained in the image document.

¬†

ImageDisplay **ImageDocumentAddImageDisplay**(ImageDocument *imgDoc*, ImageReference *image*, Number *displayType*)

Adds the given image and an image display for it of the given type.

¬†

void **ImageDocumentAddToUserInterface**(ImageDocument *imgDoc*)

Places the image document in the list of user interface documents.

¬†

void **ImageDocumentClean**(ImageDocument *imgDoc*)

Marks the image document as clean (doesn't need to be saved).

¬†

ImageDocument **ImageDocumentClone**(ImageDocument *imgDoc*, Boolean *doDeepCopy*)

Returns a duplicate of the image docuemnt, creating a copy of its images if 'doDeepCopy' is true.

¬†

void **ImageDocumentClose**(ImageDocument *imgDoc*, Boolean *saving*)

Closes the given image document. If saving is true then asks whether to save it, otherwise just closes it.

¬†

Number **ImageDocumentCountImages**(ImageDocument *imgDoc*)

Returns the number of images contained in this image document.

¬†

ImageReference **ImageDocumentCreateRGBImageFromDocument**(ImageDocument *imgDoc*, Number *width*, Number *height*, Number *extract_style*, Number *constraints*)

Creates an image by scaling the image document into (width, height).

¬†

void **ImageDocumentDeleteImage**(ImageDocument *imgDoc*, ImageReference *image*)

Deletes the given image from this image document.

¬†

Boolean **ImageDocumentDoesImageWithIDExist**(ImageDocument *imgDoc*, Number *id*)

Determines whether the image with the given id exists within this image document.

¬†

void **ImageDocumentEnsurePlacedOnPage**(ImageDocument *imgDoc*)

Makes sure the document has been layed out on the physical page.

¬†

Number **ImageDocumentGetAsPICT**(ImageDocument *imgDoc*)

Returns this image as a PICT.

¬†

Component **ImageDocumentGetComponentByID**(ImageDocument *imgDoc*, Number *id*)

Returns an annotation contained in this image document by id.

¬†

Number **ImageDocumentGetID**(ImageDocument *imgDoc*)

Gets the id of the image document.

¬†

ImageReference **ImageDocumentGetImage**(ImageDocument *imgDoc*, Number *position*)

Returns the image contained within this image document by position.

¬†

ImageReference **ImageDocumentGetImageByID**(ImageDocument *imgDoc*, Number *id*)

Returns an image contained in this image document by id.

¬†

ImageDisplay **ImageDocumentGetImageModeDisplay**(ImageDocument *imgDoc*)

Gets the image display targeted by the current image mode.

¬†

void **ImageDocumentGetMinimumPointSize**(ImageDocument *imgDoc*, NumberVariable *x*, NumberVariable *y*)

Gets the size of the minimum point in view coordinates.

¬†

String **ImageDocumentGetName**(ImageDocument *imgDoc*)

Returns the name of the image document.

¬†

void **ImageDocumentGetPageBounds**(ImageDocument *imgDoc*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)

Gets the page bounds of the document in page coordinates.

¬†

void **ImageDocumentGetPageResolution_72dpi**(ImageDocument *imgDoc*, NumberVariable *horz*, NumberVariable *vert*)

Returns the resolution of page coordinates in 72 dots per inch (returns page units per dot).

¬†

`void` **`ImageDocumentGetPageResolution_Printer`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`horz`*`, NumberVariable` *`vert`* `)`
Returns the resolution of page coordinates in printer pixels (returns page units per printer pixel).
¬†

`void` **`ImageDocumentGetPaperBounds`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the paper bounds of the document in page coordinates.
¬†

`void` **`ImageDocumentGetPreferredViewRect`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets rectangle in view coordinates of the area that is by default displayed in this mode.
¬†

`void` **`ImageDocumentGetReferencePointSize`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`x`*`, NumberVariable` *`y`* `)`
Gets the size of the reference point in view coordinates.
¬†

`Component` **`ImageDocumentGetRootComponent`**`( ImageDocument` *`imgDoc`* `)`
Gets the root annotation of the image document.
¬†

`TagGroup` **`ImageDocumentGetTagGroup`**`( ImageDocument` *`imgDoc`* `)`
Gets the tag group associated with the image document.
¬†

`void` **`ImageDocumentGetUnzoomedPointSize`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`x`*`, NumberVariable` *`y`* `)`
Gets the size of the unzoomed point in view coordinates.
¬†

`void` **`ImageDocumentGetViewExtent`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the extent in view coordinates of the items visible in the current view mode.
¬†

`void` **`ImageDocumentGetViewToWindowTransform`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`off_x`*`, NumberVariable` *`off_y`*`, NumberVariable` *`scale_x`*`, NumberVariable` *`scale_y`* `)`
Returns the transformation from view to screen coordinates.
¬†

`void` **`ImageDocumentGetVisibleViewRect`**`( ImageDocument` *`imgDoc`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the view coordinates of the rectangle visible in the view.
¬†

`DocumentWindow` **`ImageDocumentGetWindow`**`( ImageDocument` *`imgDoc`* `)`
Returns the window displaying the document.
¬†

`Boolean` **`ImageDocumentHasBeenPlacedOnPage`**`( ImageDocument` *`imgDoc`* `)`
Returns 'true' if the document has been layed out within the physical page.
¬†

`void` **`ImageDocumentHide`**`( ImageDocument` *`imgDoc`* `)`
Hides the given image document.
¬†

`Boolean` **`ImageDocumentIsInImageMode`**`( ImageDocument` *`imgDoc`* `)`
Returns true if the view of the document is in image mode.
¬†

`Boolean` **`ImageDocumentIsInPageMode`**`( ImageDocument` *`imgDoc`* `)`
Returns true if the view of the document is in page mode.
¬†

`Boolean` **`ImageDocumentIsValid`**`( ImageDocument` *`imgDoc`* `)`
Returns true if 'imageDocument' points to a valid object.
¬†

`void` **`ImageDocumentMaximizeRectInView`**`( ImageDocument` *`imgDoc`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Zooms the view so the rectangle is centered and maximal.
¬†

`Boolean` **`ImageDocumentPrint`**`( ImageDocument` *`imgDoc`* `)`
Print the image document, returning 'true' if successful.
¬†

`void` **`ImageDocumentRemoveFromUserInterface`**`( ImageDocument` *`imgDoc`* `)`
Removes the image document from the list of user interface documents.
¬†

`void` **`ImageDocumentSaveToFile`**`( ImageDocument` *`imgDoc`*`, String` *`handler`*`, String` *`fileName`* `)`
Saves the image document to the given file name using the I/O handler specified.

¬†

`void` **`ImageDocumentSetCurrentViewAsUnzoomed`**`( ImageDocument` *`imgDoc`* `)`

Makes the current view the unzoomed view.

¬†

`void` **`ImageDocumentSetName`**`( ImageDocument` *`imgDoc`*`, String` *`name`* `)`

Sets the name of the image document.

¬†

`void` **`ImageDocumentSetRectInView`**`( ImageDocument` *`imgDoc`*`, Number` *`v_t`*`, Number` *`v_l`*`, Number` *`v_b`*`, Number` *`v_r`*`, Number` *`w_t`*`, Number` *`w_l`*`, Number` *`w_b`*`, Number` *`w_r`* `)`

Zooms the view so the view rect (v_l,v_t,v_b,v_r) is displayed in the window rect (w_l,w_t,w_b,w_r).

¬†

`DocumentWindow` **`ImageDocumentShow`**`( ImageDocument` *`imgDoc`* `)`

Shows the given image document.

¬†

`DocumentWindow` **`ImageDocumentShowAtPosition`**`( ImageDocument` *`imgDoc`*`, Number` *`x`*`, Number` *`y`* `)`

Shows the given image document at the application position (x,y).

¬†

`DocumentWindow` **`ImageDocumentShowAtRect`**`( ImageDocument` *`imgDoc`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`

Shows the given image document at the rect (top,left,bottom,right).

¬†

`void` **`ImageDocumentSwitchToImageMode`**`( ImageDocument` *`imgDoc`*`, ImageDisplay` *`imgDisp`* `)`

Switches the view of the document to image mode focused on the display 'imgDisp'.

¬†

`void` **`ImageDocumentSwitchToPageMode`**`( ImageDocument` *`imgDoc`* `)`

Switches the view of the document to page mode.

¬†

`void` **`ImageDocumentUpdateDisplay`**`( ImageDocument` *`imgDoc`* `)`

Updates the display of the image document.

¬†

Related Functions

`ImageDocument` **`=`**`( ImageDocument, ! )`

¬†

`ImageDocument` **`=`**`( ImageDocument, ImageDocument )`

¬†

`ImageDocument` **`CreateImageDocument`**`( String` *`title`* `)`

Creates an empty image document.

¬†

`ImageDocument` **`GetFrontImageDocument`**`( )`

Returns the front image document.

¬†

`ImageDocument` **`GetImageDocument`**`( Number` *`position`* `)`

Returns the image document by position with the application.

¬†

`ImageDocument` **`GetImageDocumentByID`**`( Number` *`id`* `)`

Returns the image document whose id is 'id'.

¬†

`ImageDocument` **`ImageDocumentNullify`**`( ! )`

¬†

`ImageDocument` **`ImageWindowGetImageDocument`**`( DocumentWindow` *`window`* `)`

Gets the image document displayed in the window.

¬†

`ImageDocument` **`NewImageDocument`**`( String` *`title`* `)`

Creates an empty image document.

¬†

`ImageDocument` **`NewImageDocumentFromFile`**`( String` *`path_name`* `)`

Creates a new image document from a file.

¬†

Class RegionOfInterest

Return to Top

Methods

`void` **`ROIAddToMask`**`( ROI` *`roi`*`, ImageReference` *`mask`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Add the region of interest to the image within the bounds of the specified rectangle.
¬†

`void` **`ROIAddVertex`**`( ROI` *`roi`*`, Number` *`x`*`, Number` *`y`*` )`
Add a vertex with the given coordinates to the region of interest.
¬†

`void` **`ROIClearVertices`**`( ROI` *`roi`*` )`
Remove all vertices from the region of interest.
¬†

`ROI` **`ROIClone`**`( ROI` *`roi`*` )`
Returns a clone of the roi.
¬†

`Boolean` **`ROIContainsPoint`**`( ROI` *`roi`*`, Number` *`x`*`, Number` *`y`*` )`
Returns whether the region of interest encloses the given point.
¬†

`Number` **`ROICountVertices`**`( ROI` *`roi`*` )`
Return the number of vertices comprising the region of interest.
¬†

`void` **`ROIDeleteVertex`**`( ROI` *`roi`*`, Number` *`index`*` )`
Delete the given vertex from the region of interest.
¬†

`void` **`ROIGetColor`**`( ROI` *`roi`*`, NumberVariable` *`r`*`, NumberVariable` *`g`*`, NumberVariable` *`b`*` )`
Stores the color of the region of interest into the variables. Each number will be in the range of 0 to 1.
¬†

`Boolean` **`ROIGetDeletable`**`( ROI` *`roi`*` )`
Return whether the region of interest is deletable or not.
¬†

`Number` **`ROIGetID`**`( ROI` *`roi`*` )`
Return the ID for the region of interest.
¬†

`String` **`ROIGetLabel`**`( ROI` *`roi`*` )`
Return the label of the region of interest.
¬†

`void` **`ROIGetLine`**`( ROI` *`roi`*`, NumberVariable` *`sx`*`, NumberVariable` *`sy`*`, NumberVariable` *`ex`*`, NumberVariable` *`ey`*` )`
Fill in the start and end points of the line represented by the region of interest.
¬†

`Boolean` **`ROIGetMoveable`**`( ROI` *`roi`*` )`
Return whether the region of interest is moveable or not.
¬†

`String` **`ROIGetName`**`( ROI` *`roi`*` )`
Return the name of the region of interest.
¬†

`void` **`ROIGetPoint`**`( ROI` *`roi`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Return the coordinates of the point represented by this region of interest.
¬†

`void` **`ROIGetRange`**`( ROI` *`roi`*`, NumberVariable` *`start`*`, NumberVariable` *`end`*` )`
Fills in the start and end columns of the range represented by the region of interest.
¬†

`void` **`ROIGetRectangle`**`( ROI` *`roi`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`*` )`
Fill in the coordinates of the rectangle represented by the region of interest.

¬†

Boolean **ROIGetResizable**(ROI *roi*)

Return whether the region of interest is resizable or not.

¬†

void **ROIGetVertex**(ROI *roi*, Number *index*, NumberVariable *x*, NumberVariable *y*)

Return the coordinates of the given vertex of the region of interest.

¬†

Boolean **ROIGetVolatile**(ROI *roi*)

Return whether the region of interest is volatile or not.

¬†

void **ROIInsertVertex**(ROI *roi*, Number *before*, Number *x*, Number *y*)

Insert a vertex with the given coordinates before the indicated vertex of the region of interest.

¬†

Boolean **ROIIsClosed**(ROI *roi*)

Returns whether the region of interest is a closed loop or not.

¬†

Boolean **ROIIsLine**(ROI *roi*)

Return whether the region of interest is a line.

¬†

Boolean **ROIIsPoint**(ROI *roi*)

Return whether the region of interest is a point.

¬†

Boolean **ROIIsRange**(ROI *roi*)

Returns whether the region of interest is a range.

¬†

Boolean **ROIIsRectangle**(ROI *roi*)

Return whether the region of interest is a rectangle.

¬†

Boolean **ROIIsValid**(ROI *roi*)

Returns 'true' if the region of interest is a valid object.

¬†

void **ROISetColor**(ROI *roi*, Number *r*, Number *g*, Number *b*)

Set the color of the region of interest. Each number should be in the range of 0 to 1.

¬†

void **ROISetDeletable**(ROI *roi*, Boolean *deletable*)

Sets whether the region of interest should be deletable or not.

¬†

void **ROISetIsClosed**(ROI *roi*, Boolean *is_closed*)

Sets whether the region of interest is a closed loop or not (that is the last vertex connects to the first).

¬†

void **ROISetLabel**(ROI *roi*, String *name*)

Set the label on the region of interest.

¬†

void **ROISetLine**(ROI *roi*, Number *sx*, Number *sy*, Number *ex*, Number *ey*)

Set the region of interest to a line with the given start and end coordinates.

¬†

void **ROISetMoveable**(ROI *roi*, Boolean *moveable*)

Sets whether the region of interest should be moveable or not.

¬†

void **ROISetName**(ROI *roi*, String *name*)

Set the name of the region of interest.

¬†

void **ROISetPoint**(ROI *roi*, Number *x*, Number *y*)

Set the region of interest to a point with the given coordinate.

¬†

void **ROISetRange**(ROI *roi*, Number *start*, Number *end*)

Sets the region of interest to a range with the given start and end columns.

¬†

void **ROISetRectangle**(ROI *roi*, Number *top*, Number *left*, Number *bottom*, Number *right*)

Set the region of interest to a rectangle with the given coordinates.

¬†

void **ROISetRegionToValue**(ROI *roi*, ComplexImage *mask*, ComplexNumber *value*, Number *top*, Number *left*, Number *bottom*, Number *right*)

Sets the area in 'mask' corresponding to the region to the value 'value'.

¬†

`void` **`ROISetRegionToValue`**`( ROI` *`roi`*`, RGBImage` *`mask`*`, RGBNumber` *`value`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Sets the area in 'mask' corresponding to the region to the value 'value'.
¬†

`void` **`ROISetRegionToValue`**`( ROI` *`roi`*`, ImageReference` *`mask`*`, Number` *`value`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Sets the area in 'mask' corresponding to the region to the value 'value'.
¬†

`void` **`ROISetResizable`**`( ROI` *`roi`*`, Boolean` *`resizable`*` )`
Sets whether the region of interest should be resizable or not.
¬†

`void` **`ROISetVertex`**`( ROI` *`roi`*`, Number` *`index`*`, Number` *`x`*`, Number` *`y`*` )`
Set the coordinates of the given vertex of the region of interest.
¬†

`void` **`ROISetVolatile`**`( ROI` *`roi`*`, Boolean` *`is_volatile`*` )`
Set whether the region of interest is volatile or not.
¬†

Related Functions

`ROI` **`=`**`( ROI, ! )`
Assign a region of interest to a new variable.
¬†

`ROI` **`=`**`( ROI, ROI )`
Assign a region of interest to a new variable.
¬†

`ROI` **`CreateROI`**`( )`
Creates an empty region of interest.
¬†

`ROI` **`GetROIFromID`**`( Number` *`id`*` )`
Returns the region of interest associated with the ID or NULL if it does not exist.
¬†

`ROI` **`ImageDisplayGetROI`**`( ImageDisplay` *`imgDisp`*`, Number` *`index`*` )`
Returns the given ROI on this image display.
¬†

`ROI` **`ImageDisplayLookupROI`**`( ImageDisplay` *`imgDisp`*`, String` *`name`*` )`
Returns the given ROI on this image display.
¬†

`ROI` **`ImageDisplayLookupROIByID`**`( ImageDisplay` *`imgDisp`*`, Number` *`id`*` )`
Returns the ROI with the given id on this image display.
¬†

`ROI` **`NewROI`**`( )`
Creates an empty region of interest.
¬†

`ROI` **`ROINullify`**`( ! )`
¬†

Class Image

Return to Top

Methods

`void` **`ImageCalculateHistogram`**`( ImageReference` *`image`*`, ImageReference` *`hist_image`*`, Number` *`complexMode`*`, Number` *`min`*`, Number` *`max`*` )`

Calculates the histogram of 'image', mapping [min,max] into 'hist_image'.
¬†
`void` **`ImageCalculateMinMax`**`( ImageReference` *`image`*`, Number` *`surveyTechnique`*`, Number` *`complexMode`*`, NumberVariable` *`min`*`, NumberVariable` *`max`*` )`
Calculates the minimum and maximum value of 'image' using survey technique 'surveyTechnique'.
¬†
`ImageReference` **`ImageClone`**`( ImageReference` *`img`*` )`
Returns a clone of 'img'.
¬†
`void` **`ImageCopyCalibrationFrom`**`( ImageReference` *`image`*`, ImageReference` *`src_image`*` )`
Copy the calibration of 'src_image' to 'image'.
¬†
`Number` **`ImageCountImageDisplays`**`( ImageReference )`
Returns the number of image displays in which this image is displayed.
¬†
`Number` **`ImageCountImageDisplaysInImageDocument`**`( ImageReference, ImageDocument` *`imgDoc`*` )`
Returns the number of image displays in the image document in which this image is displayed.
¬†
`ImageDisplay` **`ImageCreateImageDisplay`**`( ImageReference, Number` *`displayType`*` )`
Creates a new image display of type 'displayType' for the image.
¬†
`Number` **`ImageGetDataElementBitSize`**`( ImageReference` *`img`*` )`
Returns the size of the data elements in bits.
¬†
`Number` **`ImageGetDataElementByteSize`**`( ImageReference` *`img`*` )`
Returns the smallest number of bytes that can hold a data element.
¬†
`Number` **`ImageGetDataSeed`**`( ImageReference` *`img`*` )`
Gets the seed of the image data.
¬†
`Number` **`ImageGetDataType`**`( ImageReference` *`img`*` )`
Returns a long representing the data type.
¬†
`String` **`ImageGetDescriptionText`**`( ImageReference` *`img`*` )`
Gets the description text associated with the image.
¬†
`void` **`ImageGetDimensionCalibration`**`( ImageReference, Number` *`dimension`*`, NumberVariable` *`origin`*`, NumberVariable` *`scale`*`, String` *`units`*`, Number` *`calibrationFormat`*` )`
Gets the calibration information of the given dimension.
¬†
`Number` **`ImageGetDimensionOrigin`**`( ImageReference, Number` *`dimension`*` )`
Returns the origin of the given dimension of image.
¬†
`Number` **`ImageGetDimensionScale`**`( ImageReference, Number` *`dimension`*` )`
Returns the scale of the given dimension of image.
¬†
`Number` **`ImageGetDimensionSize`**`( ImageReference, Number` *`dimension`*` )`
Gets the size of the given dimension.
¬†
`void` **`ImageGetDimensionUnitInfo`**`( ImageReference, Number` *`dimension`*`, String` *`canon_units`*`, NumberVariable` *`power`*` )`
Copies the unit string of the given dimension of image to the buffer.
¬†
`String` **`ImageGetDimensionUnitString`**`( ImageReference, Number` *`dimension`*` )`
Copies the unit string of the given dimension of image to the buffer.
¬†
`Number` **`ImageGetID`**`( ImageReference )`
Returns a unique identifier for the image.
¬†
`ImageDisplay` **`ImageGetImageDisplay`**`( ImageReference, Number` *`index`*` )`
Returns the given image display in which this image is displayed.
¬†
`ImageDisplay` **`ImageGetImageDisplayInImageDocument`**`( ImageReference, ImageDocument` *`imgDoc`*`, Number` *`index`*` )`
Returns the given image display in the image document in which this image is displayed.

¬†
Number **ImageGetIntensityOrigin**(ImageReference)
Returns the origin of image's intensity.
¬†
Number **ImageGetIntensityScale**(ImageReference)
Returns the scale of image's intensity.
¬†
void **ImageGetIntensityUnitInfo**(ImageReference, String *canon_units*, NumberVariable *power*)
Copies the unit string of image's intensity to the buffer.
¬†
String **ImageGetIntensityUnitString**(ImageReference)
Returns the units of the image's intensity.
¬†
String **ImageGetLabel**(ImageReference *img*)
Gets the label of the image as used in scripts.
¬†
String **ImageGetName**(ImageReference *img*)
Gets the name of the image.
¬†
Number **ImageGetNumDimensions**(ImageReference)
Returns number of dimensions of the image.
¬†
ImageDocument **ImageGetOrCreateImageDocument**(ImageReference *im*)
Returns an image document containing the image, creating one if necessary.
¬†
TagGroup **ImageGetTagGroup**(ImageReference *img*)
Gets the tags associated with the image.
¬†
ScriptObject **ImageGetUniqueID**(ImageReference *image*)
Returns the unique ID for this image. This id is globally unique across sessions and locations.
¬†
Boolean **ImageIsDataTypeBinary**(ImageReference *img*)
Returns true if the data in the image is binary.
¬†
Boolean **ImageIsDataTypeComplex**(ImageReference *img*)
Returns true if the data in the image is complex.
¬†
Boolean **ImageIsDataTypeFloat**(ImageReference *img*)
Returns true if the data in the image is floating point.
¬†
Boolean **ImageIsDataTypeInteger**(ImageReference *img*)
Returns true if the data in the image is integral.
¬†
Boolean **ImageIsDataTypePackedComplex**(ImageReference *img*)
Returns true if the data in the image is complex.
¬†
Boolean **ImageIsDataTypeReal**(ImageReference *img*)
Returns true if the data in the image is real.
¬†
Boolean **ImageIsDataTypeRGB**(ImageReference *img*)
Returns true if the data in the image is rgb.
¬†
Boolean **ImageIsDataTypeSignedInteger**(ImageReference *img*)
Returns true if the data in the image is integral and signed.
¬†
Boolean **ImageIsDataTypeUnsignedInteger**(ImageReference *img*)
Returns true if the data in the image is integral and unsigned.
¬†
Boolean **ImageIsDimensionCalibrationDisplayed**(ImageReference *im*, Number *dim*)
Returns 'true' if the calibration of the 'dim'th dimension is displayed.
¬†
Boolean **ImageIsIntensityCalibrationDisplayed**(ImageReference *im*)
Returns 'true' if the calibration of the intensity is displayed.
¬†
Boolean (ImageReference *image*)

ImageIsValid
Returns true if 'image' is a valid object.
¬†

`void` **ImageReadImageDataFromStream**`( ImageReference` *image*`, ScriptObject` *stream*`, Number` *stream_endianness* `)`
Reads image data from the stream. The image data type determines how the data is read, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†

`void` **ImageSetDescriptionText**`( ImageReference` *img*`, String` *description* `)`
Sets the description text associated with the image.
¬†

`void` **ImageSetDimensionCalibration**`( ImageReference, Number` *dimenstion*`, Number` *origin*`, Number` *scale*`, String` *unitString*`, Number` *calibrationFormat* `)`
Sets the calibration for the given dimension.
¬†

`void` **ImageSetDimensionCalibrationDisplayed**`( ImageReference` *im*`, Number` *dim*`, Boolean` *do_display* `)`
Sets whether or not to display the 'dim'th dimension in calibrated units to 'do_display'.
¬†

`void` **ImageSetDimensionOrigin**`( ImageReference, Number` *dimension*`, Number` *origin* `)`
Sets the origin of the given dimension of image.
¬†

`void` **ImageSetDimensionScale**`( ImageReference, Number` *dimension*`, Number` *scale* `)`
Sets the scale of the given dimension of image.
¬†

`void` **ImageSetDimensionUnitInfo**`( ImageReference, Number` *dimension*`, String` *canon_units*`, Number` *power* `)`
Sets the unit string of the given dimension of image.
¬†

`void` **ImageSetDimensionUnitString**`( ImageReference, Number` *dimension*`, String` *units* `)`
Sets the unit string of the given dimension of image.
¬†

`void` **ImageSetIntensityCalibrationDisplayed**`( ImageReference` *im*`, Boolean` *do_display* `)`
Sets whether or not to display the intensity in calibrated units to 'do_display'.
¬†

`void` **ImageSetIntensityOrigin**`( ImageReference, Number` *origin* `)`
Sets the origin of image's intensity.
¬†

`void` **ImageSetIntensityScale**`( ImageReference, Number` *scale* `)`
Sets the scale of image's intensity.
¬†

`void` **ImageSetIntensityUnitInfo**`( ImageReference, String` *canon_units*`, Number` *power* `)`
Sets the unit string of image's intensity.
¬†

`void` **ImageSetIntensityUnitString**`( ImageReference, String` *units* `)`
Sets the unit string of image's intensity.
¬†

`void` **ImageSetName**`( ImageReference` *img*`, String` *name* `)`
Sets the name of the image.
¬†

`void` **ImageSwapImageDataByteOrder**`( ImageReference )`
Swaps the byte order for each long word in the image. ABCD become DCBA.
¬†

`void` **ImageWriteImageDataToStream**`( ImageReference` *image*`, ScriptObject` *stream*`, Number` *stream_endianness* `)`
Writes image data to the stream. The image data type determines how the data is written, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†

Class ScriptObject

Return to Top

Methods

`Number` **`ScriptObjectGetClassToken`**`( ScriptObject` *`scriptObject`*`, String` *`class_name`* `)`
Gets the token in 'scriptObject' corresponding to the class 'class_name'.
¬†
`Number` **`ScriptObjectGetID`**`( ScriptObject` *`scriptObject`* `)`
Returns a unique ID for this object. The object can be recovered by using GetScriptObjectFromID function.
¬†
`Boolean` **`ScriptObjectIsValid`**`( ScriptObject` *`scriptObject`* `)`
Returns true if 'scriptObject' references a valid object.
¬†
`Function` **`ScriptObjectLookupMethod`**`( ScriptObject` *`scriptObject`*`, Function` *`meth_abs`*`, String` *`class_name`* `)`
Returns the method of this object corresponding to the abstract method 'meth_abs' and class 'class_name'.
¬†
`Function` **`ScriptObjectLookupMethod`**`( ScriptObject` *`scriptObject`*`, Function` *`meth_abs`* `)`
Returns the specific method of this object corresponding to the abstract method 'meth_abs'.
¬†

Related Functions

`ScriptObject` **`=`**`( ScriptObject` *`scriptObject1`*`, ! )`
¬†
`ScriptObject` **`=`**`( ScriptObject` *`scriptObject1`*`, ScriptObject` *`scriptObject2`* `)`
¬†
`ScriptObject` **`alloc`**`( String` *`class_name`* `)`
¬†
`ScriptObject` **`ClassNewObject`**`( String` *`class_name`* `)`
¬†
`ScriptObject` **`GetScriptObjectFromID`**`( Number` *`id`* `)`
Returns the script object associated with the ID or NULL if the object does not exist.
¬†
`ScriptObject` **`NewStreamFromBuffer`**`( Number` *`initial_size`* `)`
Creates a new stream object from from a memory buffer with initial size 'initial_size'.
¬†
`ScriptObject` **`NewStreamFromFileReference`**`( Number` *`file_ref`*`, Boolean` *`do_close`* `)`
Creates a new stream object from the file reference. On destruction, closes the file reference if 'do_close' is true.
¬†
`void` **`RegisterScriptPalette`**`( ScriptObject, String` *`type`*`, String` *`name`* `)`
¬†
`ScriptObject` **`ScriptObjectNullify`**`( ! )`
¬†
`Number` **`StreamGetPos`**`( ScriptObject` *`stream`* `)`
Returns the current position within the file.
¬†
`Number` **`StreamGetSize`**`( ScriptObject` *`stream`* `)`
Returns the size of the file.
¬†
`String` **`StreamReadAsText`**`( ScriptObject` *`stream`*`, Number` *`encoding`*`, Number` *`count`* `)`
Reads count bytes from a file, returning them as a string assuming they have the specified encoding.
¬†
`Boolean` **`StreamReadTextLine`**`( ScriptObject` *`stream`*`, Number` *`encoding`*`, String` *`str_out`* `)`
Read a line of text from the stream, storing it into the string variable, assuming the text has the specified encoding. Return 1 if successful and 0 otherwise.
¬†
`void` **`StreamSetPos`**`( ScriptObject` *`stream`*`, Number` *`base`*`, Number` *`offset`* `)`
Sets the current position within the file as an offset from 'base', where base == 0 denotes beginning of file, 1 denotes current position, and 2 denotes end of file.

¬†
`void` **`StreamSetSize`**`( ScriptObject` *`stream`*`, Number` *`size`* `)`
Sets the size of the file.
¬†
`void` **`StreamWriteAsText`**`( ScriptObject` *`stream`*`, Number` *`encoding`*`, String` *`data`* `)`
Write the string to the file with the specified encoding.
¬†

Class String

Return to Top

Methods

`String` **`StringAppend`**`( String` *`s1`*`, String` *`s2`* `)`
Appends string s2 to s1, converting its encoding to that of s1 if necessary.
¬†
`String` **`StringAppend`**`( String` *`s1`*`, Number` *`ch`*`, Number` *`encoding_id`* `)`
Appends character ch to s1, converting its encoding to that of s1 if necessary.
¬†
`Number` **`StringCompare`**`( String` *`s1`*`, String` *`s2`* `)`
Compares strings 's1' and 's2', returning -1,0, or 1 if s1 is less, equal, or greater than s2.
¬†
`String` **`StringConvertToEncoding`**`( String` *`s1`*`, Number` *`encoding_id`* `)`
Converts 's1' to the encoding specified by 'encoding_id'.
¬†
`Boolean` **`StringIsValid`**`( String` *`str`* `)`
Returns true if 'str' is a valid object.
¬†

Non-Method Script Functions

Return to Top

`RealNumberExpression` **`!`**`( RealNumberExpression` *`src`* `)`
Return the logical NOT of src.
¬†
`Boolean` **`!=`**`( String, String )`
Returns true if the two strings are not equal character-by-character.
¬†
`RealNumberExpression` **`!=`**`( RGBNumberExpression, RGBNumberExpression )`
Return 1 if the two RGB values are not equal and 0 otherwise.
¬†
`RealNumberExpression` **`!=`**`( ComplexNumberExpression, ComplexNumberExpression )`
Return 1 if the two numbers are not equal and 0 otherwise.
¬†
`RealNumberExpression` **`!=`**`( RealNumberExpression, RealNumberExpression )`
Return 1 if the two numbers are not equal and 0 otherwise.

¬†

`RealNumberExpression &&( RealNumberExpression, RealNumberExpression )`

Return the logical AND of the two numbers.

¬†

`RealNumberExpression *( RealNumberExpression, RealNumberExpression )`

Return the product of the two numbers.

¬†

`ComplexNumberExpression *( ComplexNumberExpression, ComplexNumberExpression )`

Return the product of the two numbers.

¬†

`RGBNumberExpression *( RGBNumberExpression, RGBNumberExpression )`

Return the product of the two RGB values.

¬†

`ComplexNumberExpression **( ComplexNumberExpression` *`n`*`, ComplexNumberExpression` *`exp`* `)`

Return n raised to the exp power.

¬†

`RealNumberExpression **( RealNumberExpression` *`n`*`, RealNumberExpression` *`exp`* `)`

Return n raised to the exp power.

¬†

`RealNumberExpression *=( lvalue RealNumberExpression` *`dst`*`, RealNumberExpression` *`src`* `)`

Assign dst*src to dst.

¬†

`ComplexNumberExpression *=( lvalue ComplexNumberExpression` *`dst`*`, ComplexNumberExpression` *`src`* `)`

Assign dst*src to dst.

¬†

`String +( ComplexNumber, String )`

Concatenates the string to the complex number and returns the resulting string.

¬†

`String +( String, RGBNumber )`

Concatenates the RGB value to the string and returns the resulting string.

¬†

`String +( RGBNumber, String )`

Concatenates the string to the RGB value and returns the resulting string.

¬†

`String +( String, String )`

Concatenates the string to the other string and returns the resulting string.

¬†

`String +( String, ComplexNumber )`

Concatenates the complex number to the string and returns the resulting string.

¬†

`String +( Number, String )`

Concatenates the string to the number and returns the resulting string.

¬†

`RealNumberExpression +( RealNumberExpression, RealNumberExpression )`

Return the sum of the two numbers.

¬†

`ComplexNumberExpression +( ComplexNumberExpression, ComplexNumberExpression )`

Return the sum of the two numbers.

¬†

`RGBNumberExpression +( RGBNumberExpression, RGBNumberExpression )`

Return the sum of the two RGB values.

¬†

`String +( String, Number )`

Concatenates the number to the string and returns the resulting string.

¬†

`RealNumberExpression ++( lvalue RealNumberExpression` *`dst`* `)`

Increment dst by 1.

¬†

`RealNumberExpression ++!( lvalue RealNumberExpression` *`dst`* `)`

Increment dst by 1.

¬†

`RealNumberExpression +=( lvalue RealNumberExpression` *`dst`*`, RealNumberExpression` *`src`* `)`

Assign dst+src to dst.

¬†

`ComplexNumberExpression  ( lvalue ComplexNumberExpression` *`dst`*`, ComplexNumberExpression` *`src`* `)`

+=

Assign dst+src to dst.

¬†

`String` **+=**`( String` *`s1`*`, String` *`s2`*` )`

Appends string s2 to s1, converting its encoding to that of s1 if necessary.

¬†

`RealNumberExpression` **-**`( RealNumberExpression, RealNumberExpression )`

Return the difference of the two numbers.

¬†

`RealNumberExpression` **-**`( RealNumberExpression )`

Return the negation of the number.

¬†

`RGBNumberExpression` **-**`( RGBNumberExpression, RGBNumberExpression )`

Return the difference of the two RGB values.

¬†

`RGBNumberExpression` **-**`( RGBNumberExpression )`

Return the negation of the RGB value. This is the same as RGBA(255,255,255,255)-value.

¬†

`ComplexNumberExpression` **-**`( ComplexNumberExpression, ComplexNumberExpression )`

Return the difference of the two numbers.

¬†

`ComplexNumberExpression` **-**`( ComplexNumberExpression )`

Return the negation of the number.

¬†

`RealNumberExpression` **--**`( lvalue RealNumberExpression` *`dst`*` )`

Decrement dst by 1.

¬†

`RealNumberExpression` **--!**`( lvalue RealNumberExpression` *`dst`*` )`

Decrement dst by 1.

¬†

`RealNumberExpression` **-=**`( lvalue RealNumberExpression` *`dst`*`, RealNumberExpression` *`src`*` )`

Assign dst-src to dst.

¬†

`ComplexNumberExpression` **-=**`( lvalue ComplexNumberExpression` *`dst`*`, ComplexNumberExpression` *`src`*` )`

Assign dst-src to dst.

¬†

`RGBNumberExpression` **/**`( RGBNumberExpression, RGBNumberExpression )`

Return the result of dividing the two RGB values.

¬†

`ComplexNumberExpression` **/**`( ComplexNumberExpression, ComplexNumberExpression )`

Return the result of dividing the two numbers.

¬†

`RealNumberExpression` **/**`( RealNumberExpression, RealNumberExpression )`

Return the result of dividing the two numbers.

¬†

`ComplexNumberExpression` **/=**`( lvalue ComplexNumberExpression` *`dst`*`, ComplexNumberExpression` *`src`*` )`

Assign dst/src to dst.

¬†

`RealNumberExpression` **/=**`( lvalue RealNumberExpression` *`dst`*`, RealNumberExpression` *`src`*` )`

Assign dst/src to dst.

¬†

`ImageReference` **:=**`( ImageVariable` *`image`*`, ! )`

¬†

`void` **:=**`( BasicImage` *`dst`*`, BasicImage` *`src`*` )`

Assign src to dst by reference.

¬†

`RealNumberExpression` **<**`( RealNumberExpression, RealNumberExpression )`

Return 1 if the left number is less than the right number and 0 otherwise.

¬†

`RealNumberExpression` **<=**`( RealNumberExpression, RealNumberExpression )`

Return 1 if the left number is less or equal to than the right number and 0 otherwise.

¬†

`SurfacePlotImageDisplay` **=**`( SurfacePlotImageDisplay` *`dst`*`, ImageDisplay` *`id`*` )`

¬†

`SurfacePlotImageDisplay` **=**`( SurfacePlotImageDisplay` *`dst`*`, ! )`

¬†

`SurfacePlotImageDisplay =( SurfacePlotImageDisplay` *`dst`*`, SurfacePlotImageDisplay` *`spid`* `)`

¬†

`DocumentWindow =( DocumentWindow, ! )`

¬†

`DocumentWindow =( DocumentWindow, DocumentWindow )`

¬†

`ImageDocument =( ImageDocument, ! )`

¬†

`ImageDocument =( ImageDocument, ImageDocument )`

¬†

`ImageDisplay =( ImageDisplay, Component )`

¬†

`ImageDisplay =( ImageDisplay, ! )`

¬†

`ImageDisplay =( ImageDisplay, ImageDisplay )`

¬†

`Component =( Component, ! )`

¬†

`Component =( Component, Component )`

¬†

`RGBNumberExpression =( lvalue RGBNumberExpression` *`dst`*`, RGBNumberExpression` *`src`* `)`

Assign src to dst.

¬†

`ComplexNumberExpression =( lvalue ComplexNumberExpression` *`dst`*`, ComplexNumberExpression` *`src`* `)`

Assign src to dst.

¬†

`RealNumberExpression =( lvalue RealNumberExpression` *`dst`*`, RealNumberExpression` *`src`* `)`

Assign src to dst.

¬†

`LinePlotImageDisplay =( LinePlotImageDisplay` *`dst`*`, LinePlotImageDisplay` *`lpid`* `)`

¬†

`LinePlotImageDisplay =( LinePlotImageDisplay` *`dst`*`, ! )`

¬†

`String =( String` *`dst`*`, ! )`

Assigns NULL to dst string.

¬†

`String =( String` *`dst`*`, String` *`src`* `)`

Assigns src string to dst string.

¬†

`Throwable =( Throwable` *`dst`*`, ! )`

Assigns NULL to dst throwable.

¬†

`Throwable =( Throwable` *`dst`*`, Throwable` *`src`* `)`

Assigns src throwable to dst throwable.

¬†

`Function =( Function` *`dst`*`, ! )`

Assigns NULL to dst function.

¬†

`Function =( Function` *`dst`*`, Function` *`src`* `)`

Assigns src function to dst function.

¬†

`ScriptObject =( ScriptObject` *`scriptObject1`*`, ! )`

¬†

`ScriptObject =( ScriptObject` *`scriptObject1`*`, ScriptObject` *`scriptObject2`* `)`

¬†

`ROI =( ROI, ! )`

Assign a region of interest to a new variable.

¬†

`ROI =( ROI, ROI )`

Assign a region of interest to a new variable.

¬†

`RasterImageDisplay =( RasterImageDisplay` *`dst`*`, ! )`

¬†

`RasterImageDisplay =( RasterImageDisplay` *`dst`*`, RasterImageDisplay` *`rid`* `)`

¬†

TagGroup **=**(TagGroup *tagGroup1*, !)
¬†
TagGroup **=**(TagGroup *tagGroup1*, TagGroup *tagGroup2*)
¬†
LinePlotImageDisplay **=**(LinePlotImageDisplay *dst*, ImageDisplay *id*)
¬†
RasterImageDisplay **=**(RasterImageDisplay *dst*, ImageDisplay *id*)
¬†
RealNumberExpression **==**(RGBNumberExpression, RGBNumberExpression)
Return 1 if the left RGB value is equal to the right RGB value and 0 otherwise.
¬†
RealNumberExpression **==**(ComplexNumberExpression, ComplexNumberExpression)
Return 1 if the left number is equal to the right number and 0 otherwise.
¬†
RealNumberExpression **==**(RealNumberExpression, RealNumberExpression)
Return 1 if the left number is equal to the right number and 0 otherwise.
¬†
Boolean **==**(String, String)
Returns true if the two strings are equal character-by-character.
¬†
RealNumberExpression **>**(RealNumberExpression, RealNumberExpression)
Return 1 if the left number is greater than the right number and 0 otherwise.
¬†
RealNumberExpression **>=**(RealNumberExpression, RealNumberExpression)
Return 1 if the left number is greater than or equal to the right number and 0 otherwise.
¬†
RealNumberExpression **?**(RealNumberExpression *condition*, RealNumberExpression *truenumber*, RealNumberExpression *falsenumber*)
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
ComplexNumberExpression **?**(RealNumberExpression *condition*, ComplexNumberExpression *truenumber*, ComplexNumberExpression *falsenumber*)
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
RGBNumberExpression **?**(RealNumberExpression *condition*, RGBNumberExpression *truenumber*, RGBNumberExpression *falsenumber*)
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
Number **[**(String, Number *index*)
Returns the numeric value in the string's encoding of the first character of the string.
¬†
BasicImage **[**(BasicImage, Number *x*, Number *y*)
Returns a pixel in the given real image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†
BasicImage **[**(BasicImage, Number *x*, Number *y*, Number *z*)
Returns a pixel in the given real image at the position [x,y,z]. This operator is equivalent to Index(image,x,y,z).
¬†
BasicImage **[**(BasicImage, Number *top*, Number *left*, Number *bottom*, Number *right*)
Returns the sub-area image specified by the coordinates [top,left,bottom,right].
¬†
BasicImage **[**(BasicImage, Number *y1*, Number *x1*, Number *z1*, Number *y2*, Number *x2*, Number *z2*)
Returns the sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†
RealImage **[**(RealImage, Number *x*, Number *y*)
Returns a pixel in the given real image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†
RealImage **[**(RealImage, Number *x*, Number *y*, Number *z*)
Returns a pixel in the given real image at the position [x,y,z]. This operator is equivalent to Index(image,x,y,z).
¬†
RealImage **[**(RealImage, Number *top*, Number *left*, Number *bottom*, Number *right*)
Returns the sub-area image specified by the coordinates [top,left,bottom,right].
¬†
RealImage **[**(RealImage, Number *y1*, Number *x1*, Number *z1*, Number *y2*, Number *x2*, Number *z2*)
Returns the sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†

ComplexImage **[**(ComplexImage, Number *top*, Number *left*, Number *bottom*, Number *right*)
Returns the complex sub-area image specified by the coordinates [top,left,bottom,right].
¬†

ComplexImage **[**(ComplexImage, Number *y1*, Number *x1*, Number *z1*, Number *y2*, Number *x2*, Number *z2*)
Returns the complex sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†

ComplexImage **[**(ComplexImage, Number *x*, Number *y*)
Returns a complex pixel in the given complex image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†

RGBImage **[**(RGBImage, Number *x*, Number *y*)
Returns a RGB pixel in the given RGB image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†

RGBImage **[**(RGBImage, Number *top*, Number *left*, Number *bottom*, Number *right*)
Returns the RGB sub-area image specified by the coordinates [top,left,bottom,right].
¬†

RGBImage **[**(RGBImage, Number *y1*, Number *x1*, Number *z1*, Number *y2*, Number *x2*, Number *z2*)
Returns the RGB sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†

RealNumberExpression **||**(RealNumberExpression, RealNumberExpression)
Return the logical OR of the two numbers.
¬†

void **AbortAcquisitionDaemon**(ImageReference)
Abort all acquisition daemons associated with the image.
¬†

void **AbortAllAcquisitionDaemons**( )
Abort all acquisition daemons with any image within the application.
¬†

RealNumberExpression **abs**(ComplexNumberExpression)
Return the absolute value of the modulus of a complex number.
¬†

RealNumberExpression **abs**(RealNumberExpression)
Return the absolute value of the number.
¬†

RealNumberExpression **acos**(RealNumberExpression)
Return the arc-cosine of the number.
¬†

void **AddArrayMaskToImage**(ImageReference *mask*, Number *x_center*, Number *y_center*, Number *x_radius*, Number *y_radius*, Number *x_v1*, Number *y_v1*, Number *x_v2*, Number *y_v2*, Number *filter_length*, Boolean *do_inverse*)
¬†

void **AddBandPassMaskToImage**(ImageReference *mask*, Number *x_center*, Number *y_center*, Number *band_radius_1*, Number *band_radius_2*, Number *filter_length*, Boolean *do_inverse*)
¬†

void **AddOvalMaskToImage**(ImageReference *mask*, Number *t_oval*, Number *l_oval*, Number *b_oval*, Number *r_oval*, Number *filter_length*, Boolean *do_inverse*)
¬†

void **AddPathToCopyToImageList**(String *path*)
Adds 'path' to the list of tags to transfer to acquired images.
¬†

void **AddScriptFileToMenu**(String *fileName*, String *commandName*, String *menuName*, String *optionalSubMenuName*, Boolean *isLibrary*)
Install the script in the file indicated by fileName into the application. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.
¬†

void **AddScriptFileToPackage**(String *fileName*, String *packageName*, Number *packageLevel*, String *commandName*, String *menuName*, String *optionalSubMenuName*, Boolean *isLibrary*)
Install the script in the file indicated by fileName into the package. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.
¬†

void **AddScriptToMenu**(String *script*, String *commandName*, String *menuName*, String *optionalSubMenuName*, Boolean *isLibrary*)
Install the script into the application. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.

¬†

`void` **AddScriptToPackage**(String *script*, String *packageName*, Number *packageLevel*, String *commandName*, String *menuName*, String *optionalSubMenuName*, Boolean *isLibrary*)

Install the script into the application. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.

¬†

`void` **AddStringToList**(ImageReference, String *tagPath*, String *string*)

Adds the string to the image tag list indicated by tagPath.

¬†

`void` **AddStringToPersistentList**(String *tagPath*, String *string*)

Adds the string to the persistent tag list indicated by tagPath.

¬†

`void` **AddTagsToPackage**(TagGroup *tags*, String *packageName*, Number *packageLevel*, String *identifier*)

Install the tags into the package. The identifier is used to identify the tags in the packages. Clients should take care to use unique identifiers. See the Java model of naming classes.

¬†

`void` **AddTwinMaskToImage**(ImageReference *mask*, Number *x_center*, Number *y_center*, Number *x_offset*, Number *y_offset*, Number *x_radius*, Number *y_radius*, Number *filter_length*, Boolean *do_inverse*)

¬†

`void` **AddWedgeMaskToImage**(ImageReference *mask*, Number *x_center*, Number *y_center*, Number *x_v1*, Number *y_v1*, Number *x_v2*, Number *y_v2*, Number *filter_length*, Boolean *do_inverse*)

¬†

`void` **AdjustScriptMenuItem**(String *commandName*, String *menuName*, String *optionalSubMenuName*, String *newCommandName*, Boolean *enabled*, Boolean *checked*, Number *key*, Number *acceleratorPos*)

Adjusts the display characteristics of the given script menu item. NewCommandName specifies the new name for the menu item. The menu item will have to be referred to by that name from then forth. Enabled/checked indicate whether the item is enabled/checked. Key refers to the command key equivalent on the MacOS. Pass 0 to have no command key equivalent. AcceleratorPos refers to the position of the accelerator equivalent on Windows. Pass -1 to have to accelerator.

¬†

`RealNumberExpression` **AiryAi**(Number)

Return the Airy Ai function of the number.

¬†

`RealNumberExpression` **AiryBi**(Number)

Return the Airy Bi function of the number.

¬†

`ScriptObject` **alloc**(String *class_name*)

¬†

`RealNumberExpression` **alpha**(RGBNumberExpression)

Return the alpha portion of an RGB number.

¬†

`Number` **AnnotationType**(ImageReference, Number *annotationID*)

Return the annotation type indicated by the annotation ID within the image.

¬†

`void` **ApplicationGetBounds**(NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)

Gets the bounds of the main area of the application in application coordinates.

¬†

`void` **ApplicationGetOrigin**(NumberVariable *x*, NumberVariable *y*)

Gets the origin of the application in global coordinates.

¬†

`void` **ApplyDataBar**(ImageDisplay *imgDisp*)

Applies a data bar to the image.

¬†

`Number` **asc**(String, Number *index*)

Returns numeric value of he 'index'th character in of the string.

¬†

`Number` **asc**(String)

Returns the numeric value in ascii of the first character of the string.

¬†

`RealNumberExpression` **asin**(RealNumberExpression)

Return the arc-sine of the number.

¬†

`RealNumberExpression` **atan**(RealNumberExpression)

Return the arc-tangent of the number.

¬†

RealNumberExpression **atan2**(RealNumberExpression *y*, RealNumberExpression *x*)
Return the arc-tangent of the ratio of y/x.
¬†
RealNumberExpression **atanh**(RealNumberExpression)
Return the hyperbolic arc-tangent of the number.
¬†
RealImage **AutoCorrelate**(RealImage *source*)
Return an image which is the result of the auto correlation of source.
¬†
RealImage **AutoCorrelation**(RealImage *source*)
Return an image which is the result of the auto correlation of source.
¬†
Number **average**(RealImageExpression *image*)
Return the average pixel value of the image expression.
¬†
String **BaseN**(Number *n*, Number *base*)
Returns the number as a base n string.
¬†
String **BaseN**(Number *n*, Number *base*, Number *length*)
Returns the number as a base n string of the given length.
¬†
void **Beep**( )
Play the current system beep.
¬†
RealNumberExpression **BesselI**(Number, Number)
Return the general Bessel I function of two numbers.
¬†
RealNumberExpression **BesselJ**(Number, Number)
Return the general Bessel J function of two numbers.
¬†
RealNumberExpression **BesselK**(Number, Number)
Return the general Bessel K function of two numbers.
¬†
RealNumberExpression **BesselY**(Number, Number)
Return the general Bessel Y function of two numbers.
¬†
RealNumberExpression **Beta**(Number, Number)
Return the beta function of two numbers.
¬†
String **Binary**(Number *n*, Number *length*)
Returns the number as a binary string of the given length.
¬†
String **Binary**(Number *n*)
Returns the number as a binary string.
¬†
RealImage **BinaryImage**(String *title*, Number *d0*)
Creates a 1D binary image of size [d0] with the given title.
¬†
RealImage **BinaryImage**(String *title*, Number *d0*, Number *d1*)
Creates a 2D binary image of size [d0,d1] with the given title.
¬†
RealImage **BinaryImage**(String *title*, Number *d0*, Number *d1*, Number *d2*)
Creates a 3D binary image of size [d0,d1,d2] with the given title.
¬†
RealNumberExpression **BinomialCoefficient**(Number, Number)
Return the binomial coefficient of two numbers.
¬†
RealNumberExpression **BinomialRandom**(Number, Number)
Return a random number with binomial distribution between [0,1)
¬†
RealNumberExpression **blue**(RGBNumberExpression)
Return the blue portion of an RGB number.
¬†
void **BrowseTagFile**( )

Present an open file dialog to the user, allow them to select a tag file, and then allow them to browse through it.
¬†

`RealNumberExpression` **`Ceil`**`( RealNumberExpression )`
Return the number truncated to an integer (rounding towards positive infinity).
¬†

`void` **`CheckHeap`**`( )`
Checks the integrity of the application memory.
¬†

`Boolean` **`ChooseMenuItem`**`( String` *`menu`*`, String` *`subMenu`*`, String` *`item`*` )`
Choose the given menu item.
¬†

`String` **`chr`**`( Number` *`n`*` )`
Returns the ASCII character specified by n as a string.
¬†

`ComplexNumberExpression` **`cis`**`( RealNumberExpression )`
Return complex(cos(n),sin(n)) as a complex number, where n is a real number. This is a unit vector in the complex plane.
¬†

`Function` **`ClassAddMethod`**`( String` *`class_name`*`, Function` *`method`*` )`
¬†

`ScriptObject` **`ClassNewObject`**`( String` *`class_name`*` )`
¬†

`void` **`ClassRemoveMethod`**`( String` *`class_name`*`, Function` *`method`*` )`
¬†

`void` **`CleanImage`**`( ImageReference )`
Mark the image as having been saved.
¬†

`void` **`ClearImage`**`( ImageReference )`
Set each pixel in the image to zero.
¬†

`void` **`ClearSelection`**`( ImageReference )`
Remove selection (if any) from the image.
¬†

`RealNumberExpression` **`clip`**`( RealNumberExpression` *`value`*`, RealNumberExpression` *`minimum`*`, RealNumberExpression` *`maximum`*` )`
Return the value clipped to be in the range bounded by minimum and maximum.
¬†

`Boolean` **`ClipboardGetAsPicture`**`( NumberVariable` *`picture`*` )`
Gets the contents of the clipboard as a picture, if possible, and returns true if successful.
¬†

`Boolean` **`ClipboardGetAsString`**`( String` *`str`*` )`
Gets the contents of the clipboard as a string with the given encoding, if possible, and returns true if successful.
¬†

`Boolean` **`ClipboardGetAsTagGroup`**`( TagGroup` *`tagGroup`*` )`
Gets the contents of the clipboard as a tag group, if possible, and returns true if successful.
¬†

`void` **`ClipboardSetAsPicture`**`( Number` *`picture`*` )`
Sets the contents of the clipboard to the picture.
¬†

`void` **`ClipboardSetAsString`**`( String )`
Sets the contents of the clipboard to the text.
¬†

`void` **`ClipboardSetAsTagGroup`**`( TagGroup` *`tagGroup`*` )`
Sets the contents of the clipboard to the tag group.
¬†

`void` **`CloseFile`**`( Number` *`file`*` )`
Close the file. This function should be called to close a file whenever a file is opened.
¬†

`void` **`CloseImage`**`( ImageReference )`
Attempt to close the image. If the data has changed, a dialog box appears to ask the user to save the image before closing it.
¬†

`void` **`CloseProgressWindow`**`( )`
Close the progress window if it is open.
¬†

`void` **`CloseTimeBar`**`( )`
Closes the time bar.

¬†

Boolean **CommandDown**( )

Returns 1 if the command key is down and 0 otherwise.

¬†

ComplexNumberExpression **complex**(RealNumberExpression *real*, RealNumberExpression *imaginary*)

Returns a complex number with the value of real + i * imaginary.

¬†

ComplexImage **ComplexImage**(String *title*, Number *bytes*, Number *d0*)

Creates a 1D complex image of size [d0] with the given title. The bytes parameter can be 8 or 16 for single and double precision floating point numbers.

¬†

ComplexImage **ComplexImage**(String *title*, Number *bytes*, Number *d0*, Number *d1*)

Creates a 2D complex image of size [d0,d1] with the given title. The bytes parameter can be 8 or 16 for single and double precision floating point numbers.

¬†

ComplexImage **ComplexImage**(String *title*, Number *bytes*, Number *d0*, Number *d1*, Number *d2*)

Creates a 3D complex image of size [d0,d1,d2] with the given title. The bytes parameter can be 8 or 16 for single and double precision floating point numbers.

¬†

ComplexImage **ComplexToPacked**(ComplexImage *source*)

Creates a new packed complex image from the complex 8-byte source.

¬†

ComplexImage **ComplexToPacked**(ComplexImage *source*, Number *style*)

Creates a new packed complex image from the complex 8-byte source.

¬†

Component **ComponentNullify**(!)

¬†

ComplexNumberExpression **conjugate**(ComplexNumberExpression)

Return the conjugate of the complex number.

¬†

void **ConnectObject**(Number *object*, String *message*, String *ident*, ScriptObject *scriptObject*, String *method*)

Build the connection with the given object and message under ident.

¬†

Boolean **ContinueCancelDialog**(String *prompt*)

Puts up a dialog with both a Continue button and Cancel button. Returns 1 for Continue and 0 for Cancel.

¬†

Boolean **ControlDown**( )

Returns 1 if the control key is down and 0 otherwise.

¬†

String **ConvertEndOfLine**(Number *eol_format*, String *str*)

Converts the string to the end-of-line format indicated by 'eol_format_index': 0 == platform format, 1 = Macintosh format, 2 = Windows format.

¬†

void **ConvertImageData**(ImageReference *from*, ImageReference *to*)

¬†

void **ConvertImageDataSlice_2D**(ImageReference *from*, ImageReference *to*, Number *x_1*, Number *y_1*, Number *f_x_0*, Number *f_y_0*, Number *f_x_d*, Number *f_x_s*, Number *f_y_d*, Number *f_y_s*, Number *t_x_0*, Number *t_y_0*, Number *t_x_d*, Number *t_x_s*, Number *t_y_d*, Number *t_y_s*)

¬†

void **ConvertToByte**(ImageReference)

Converts the given image to unsigned integer 1-byte data.

¬†

void **ConvertToComplex**(ImageReference)

Converts the given image to complex single precision data.

¬†

void **ConvertToFloat**(ImageReference)

Converts the given image to single precision real data.

¬†

void **ConvertToLong**(ImageReference)

Converts the given image to signed integer 4-byte data.

¬†

void **ConvertToPackedComplex**(ImageReference)

Converts the given image to packed complex data.

¬†

void **ConvertToShort**(ImageReference)
Converts the given image to signed integer 2-byte data.
¬†
RealImage **Convolution**(ImageReference *source*, ImageReference *kernel*)
Creates a new image that is the convolution of the source image with the kernel. The kernel should be less than 7x7.
¬†
ComplexNumberExpression **cos**(ComplexNumberExpression)
Return the cosine of the number
¬†
RealNumberExpression **cos**(RealNumberExpression)
Return the cosine of the number.
¬†
ComplexNumberExpression **cosh**(ComplexNumberExpression)
Return the hyperbolic cosine of the number.
¬†
RealNumberExpression **cosh**(RealNumberExpression)
Return the hyperbolic cosine of the number.
¬†
Number **CountAllImages**( )
Returns the number of images.
¬†
Number **CountAnnotations**(ImageReference)
Return the number of annotations within the image.
¬†
Number **CountDocumentWindows**( )
Returns the number of document windows.
¬†
Number **CountDocumentWindowsOfType**(Number *type*)
Returns the number of document windows with type 'type'.
¬†
Number **CountFloatingWindows**( )
Returns the number of floating windows.
¬†
Number **CountImageDocuments**( )
Returns the number of image documents.
¬†
Number **CountImages**( )
Count the number of images.
¬†
Number **CountListEntries**(ImageReference, String *tagPath*)
Returns the number of tags within the given image tag list or group indicated by tagPath.
¬†
Number **CountPersistentListEntries**(String *tagPath*)
Returns the number of tags within the given persistent tag list or group indicated by tagPath.
¬†
Number **CountScreens**( )
Returns the number of screens.
¬†
Number **CountScriptFunctions**(String *fnName*)
Returns the number of script functions having name 'fnName'.
¬†
Number **CreateArrowAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Create an arrow annotation in the image with the endpoints [top,left] and [bottom,right]. Return the new annotation's ID.
¬†
Number **CreateBoxAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Create a box annotation in the image with the coordinates [top, left, bottom, right]. Return the new annotation's ID.
¬†
RealImage **CreateByteImage**(String *title*, Number *width*, Number *height*)
Creates a 2D unsigned 1-byte integer image of size [width,height] with the given title.
¬†
ComplexImage **CreateComplexImage**(String *title*, Number *width*, Number *height*)
Creates a 2D single precision complex image of size [width,height] with the given title.
¬†
void **CreateDirectory**(String *fileName*)

Create a folder named fileName.
¬†
`Number` **`CreateDoubleArrowAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Create an double ended arrow annotation in the image with the endpoints [top,left] and [bottom,right]. Return the new annotation's ID.
¬†
`void` **`CreateFile`**`( String` *`fileName`* `)`
Create a file named fileName.
¬†
`Number` **`CreateFileForWriting`**`( String` *`fileName`* `)`
Create and open the file for writing. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.
¬†
`RealImage` **`CreateFloatImage`**`( String` *`title`*`, Number` *`width`*`, Number` *`height`* `)`
Creates a 2D single precision float image of size [width,height] with the given title.
¬†
`ImageDocument` **`CreateImageDocument`**`( String` *`title`* `)`
Creates an empty image document.
¬†
`RGBImage` **`CreateImageFromDisplay`**`( ImageReference )`
Convert the display of image to an RGB image.
¬†
`Number` **`CreateLineAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Create an line annotation in the image with the endpoints [top,left] and [bottom,right]. Return the new annotation's ID.
¬†
`RealImage` **`CreateLongImage`**`( String` *`title`*`, Number` *`width`*`, Number` *`height`* `)`
Creates a 2D signed 4-byte integer image of size [width,height] with the given title.
¬†
`ImageReference` **`CreateMaskFromAnnotations`**`( RasterImageDisplay` *`rid`*`, Number` *`filter_length`*`, Boolean` *`is_opaque`*`, NumberVariable` *`has_mask`* `)`
¬†
`Number` **`CreateOvalAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Create an oval annotation in the image with the coordinates [top, left, bottom, right]. Return the new annotation's ID.
¬†
`ComplexImage` **`CreatePackedComplexImage`**`( String` *`title`*`, Number` *`width`*`, Number` *`height`* `)`
Creates a 2D packed complex image of size [width,height] with the given title.
¬†
`Number` **`CreatePictureAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*`, !` *`PicHandle`* `)`
Create a picture annotation in the image with the coordinates [top, left, bottom, right] from the picHandle. Return the new annotation's ID.
¬†
`RGBImage` **`CreateRGBImage`**`( String` *`title`*`, Number` *`width`*`, Number` *`height`* `)`
Creates a 2D RGB image of size [width,height] with the given title.
¬†
`RGBImage` **`CreateRGBImageFromPicture`**`( Number` *`picture`* `)`
Create an RGB image by drawing into it with a picture
¬†
`ROI` **`CreateROI`**`( )`
Creates an empty region of interest.
¬†
`RealImage` **`CreateShortImage`**`( String` *`title`*`, Number` *`width`*`, Number` *`height`* `)`
Creates a 2D signed 2-byte integer image of size [width,height] with the given title.
¬†
`Number` **`CreateTextAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, String` *`text`* `)`
Create a text annotation in the image at the position [top,left] with text. Return the new annotation's ID.
¬†
`RealImage` **`CrossCorrelate`**`( RealImage` *`source1`*`, RealImage` *`source2`* `)`
Return an image which is the result of the cross correlation of source1 and source2.
¬†
`RealImage` **`CrossCorrelation`**`( RealImage` *`source1`*`, RealImage` *`source2`* `)`
Return an image which is the result of the cross correlation of source1 and source2.
¬†
`RealImage` **`CrossProduct`**`( RealImage` *`a`*`, RealImage` *`b`* `)`
Return the matrix cross product image of matrix images a and b.
¬†

`String` **`DateStamp`**`( )`
Return a string representing the current date and time.
¬†
`String` **`Decimal`**`( Number` *`n`* `)`
Returns the number as a decimal string.
¬†
`String` **`Decimal`**`( Number` *`n`*`, Number` *`length`* `)`
Returns the number as a decimal string of the given length.
¬†
`void` **`Delay`**`( Number )`
Delay for the given number of tick counts.
¬†
`void` **`DeleteAnnotation`**`( ImageReference, Number` *`annotationID`* `)`
Delete the annotation indicated by the annotationID within the image.
¬†
`void` **`DeleteAnnotationNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`* `)`
Removes the annotation note with the label noteLabel.
¬†
`void` **`DeleteDirectory`**`( String` *`dirName`* `)`
Deletes the folder named dirName.
¬†
`void` **`DeleteFile`**`( String` *`fileName`* `)`
Delete the file.
¬†
`void` **`DeleteImage`**`( ImageVariable )`
Close the image without asking the user to save it.
¬†
`void` **`DeleteImageFile`**`( String` *`fileName`* `)`
Delete the image file.
¬†
`void` **`DeleteListEntry`**`( ImageReference, String` *`tagPath`*`, Number` *`index`* `)`
Deletes the tag with the given index from within the image tag list indicated by tagPath.
¬†
`void` **`DeleteNote`**`( ImageReference, String` *`noteLabel`* `)`
Removes the image note with the label noteLabel.
¬†
`void` **`DeletePersistentListEntry`**`( String` *`tagPath`*`, Number` *`index`* `)`
Deletes the tag with the given index from within the persistent tag list indicated by tagPath.
¬†
`void` **`DeletePersistentNote`**`( String` *`noteLabel`* `)`
Removes the note with the label noteLabel from the persistent note list.
¬†
`void` **`DeselectAnnotation`**`( ImageReference, Number` *`annotationID`* `)`
Deselect the annotation indicated by the annotationID within the image.
¬†
`void` **`DestroyPicture`**`( NumberVariable` *`picture`* `)`
Destroy a picture
¬†
`void` **`DisconnectObject`**`( Number` *`object`*`, String` *`message`*`, String` *`ident`* `)`
Break the connection specified by ident for the given object and message.
¬†
`void` **`DisplayAt`**`( ImageReference, Number` *`x`*`, Number` *`y`* `)`
Display the image's image document if it is not display already and moves the window position to [x,y] screen coordinates.
¬†
`RealNumberExpression` **`distance`**`( RealNumberExpression` *`x`*`, RealNumberExpression` *`y`* `)`
Return sqrt(x * x + y * y).
¬†
`Boolean` **`DoesClassExist`**`( String` *`class_name`* `)`
¬†
`Boolean` **`DoesDirectoryExist`**`( String` *`dirName`* `)`
Returns 'true' if the named directory exists
¬†
`Boolean` **`DoesFileExist`**`( String` *`dirName`* `)`
Returns 'true' if the named file exists

¬†
Boolean **DoesFunctionExist**(String *fnName*)
Determines if the given function exists.
¬†
Boolean **DoesImageExist**(Number *imageID*)
Determine if the image with imageID exists and returns 1 if it does; return 0 otherwise.
¬†
void **DoEvents**( )
Process all pending MacOS events.
¬†
ComplexNumber **DotProduct**(ComplexImageExpression, ComplexImageExpression)
Return the dot product of two image expressions (which are treated as vectors).
¬†
Number **DotProduct**(RealImageExpression, RealImageExpression)
Return the dot product of two image expressions (which are treated as vectors).
¬†
void **EditorWindowAddText**(DocumentWindow *window*, String *text*)
Appends the text to a editor window.
¬†
void **EditorWindowGetFont**(DocumentWindow *window*, String *face_name*, NumberVariable *attributes*, NumberVariable *size*, NumberVariable *text_encoding*)
Gets the font of a script window.
¬†
String **EditorWindowGetText**(DocumentWindow *window*)
Gets the text in an editor window.
¬†
Boolean **EditorWindowPrint**(DocumentWindow *window*)
Prints the editor window.
¬†
void **EditorWindowSaveToFile**(DocumentWindow *window*, String *path*)
Saves the editor window to the specified path.
¬†
void **EditorWindowSetFont**(DocumentWindow *window*, String *face_name*, Number *attributes*, Number *size*, Number *text_encoding*)
Sets the font of a script window.
¬†
void **EditorWindowSetText**(DocumentWindow *window*, String *text*)
Sets the text in an editor window.
¬†
void **EMBeamShift**(Number *xAmount*, Number *yAmount*)
Shift the beam by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMChangeFocus**(Number *amount*)
Change the focus by amount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMChangeStigmation**(Number *xAmount*, Number *yAmount*)
Change the stigmation by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMChangeTilt**(Number *xAmount*, Number *yAmount*)
Change the tilt by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMCloseCommunication**( )
Close communication to the microscope. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMImageShift**(Number *xAmount*, Number *yAmount*)
Shift the image by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareImageShift**( )
Prepare image shift. Call this before a sequence of image shift changes. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareShift**( )
Prepare beam shift. Call this before a sequence of beam shift changes. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareStigmation**( )

Prepare stigmation. Call this before a sequence of stigmation changes. The EM Control Plug-in must be currently installed and configured.
¬†

`void` **`EMPrepareTilt`**`( )`
Prepare tilt. Call this before a sequence of tilt changes. The EM Control Plug-in must be currently installed and configured.
¬†

`void` **`EMSetupCommunication`**`( )`
Setup communication with the microscope. The EM Control Plug-in must be currently installed and configured.
¬†

`RealNumberExpression` **`erf`**`( Number )`
Return the error function of the number.
¬†

`RealNumberExpression` **`erfc`**`( Number )`
Return the complement of the error function of the number.
¬†

`void` **`ExecutePersistentScriptGroup`**`( String` *`tagPath`*`, String` *`form`* `)`
Execute a group of script functions in the persistent tag list indicated by tagPath. The actual scripts executed will be formed by sprintf'ing into the form parameter. The form parameter should contain exactly one %s in which will be inserted the function name.
¬†

`Number` **`ExecuteScriptFile`**`( String` *`fileName`*`, Number` *`script_index`* `)`
Executes the script file indicated by fileName and returns the exit value of that script. A script may specify a specific exit value by exiting with the exit(n) function. If a script does not use the exit() function the exit value will be 0.
¬†

`Number` **`ExecuteScriptFile`**`( String` *`fileName`* `)`
Executes the script file indicated by fileName and returns the exit value of that script. A script may specify a specific exit value by exiting with the exit(n) function. If a script does not use the exit() function the exit value will be 0.
¬†

`void` **`ExecuteScriptGroup`**`( ImageReference` *`image`*`, String` *`tagPath`*`, String` *`form`* `)`
Execute a group of script functions in the image tag list indicated by tagPath. The actual scripts executed will be formed by sprintf'ing into the form parameter. The form parameter should contain exactly one %s in which will be inserted the function name.
¬†

`Number` **`ExecuteScriptString`**`( String` *`text`* `)`
Executes the script text and returns the exit value of that script. A script may specify a specific exit value by exiting with the exit(n) function. If a script does not use the exit() function the exit value will be 0.
¬†

`void` **`exit`**`( Number )`
Stops execution of the current script immediately and returns the number as the result of the script.
¬†

`ComplexNumberExpression` **`exp`**`( ComplexNumberExpression )`
Return the exponential of the number.
¬†

`RealNumberExpression` **`exp`**`( RealNumberExpression )`
Return the exponential of the number.
¬†

`RealNumberExpression` **`exp1`**`( RealNumberExpression )`
Return the exponential - 1 of the number.
¬†

`RealNumberExpression` **`exp10`**`( RealNumberExpression )`
Return 10 raised to the number.
¬†

`RealNumberExpression` **`exp2`**`( RealNumberExpression )`
Return 2 raised to the number.
¬†

`RealNumberExpression` **`ExponentialRandom`**`( )`
Return a random number with exponential distribution between [0,1).
¬†

`ImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, ImageExpression` *`value`* `)`
The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.
¬†

`RGBImageExpression` **`ExprSize`**`( ImageReference` *`image`*`, RGBImageExpression` *`value`* `)`
The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to the size of image expression and this function will return the value parameter.
¬†

`RGBImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, RGBImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a RGB image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.

¬†

`ImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, ImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.

¬†

`RGBImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, RGBImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of an RGB image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.

¬†

`ComplexImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, ComplexImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a complex image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.

¬†

`ComplexImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, ComplexImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a complex image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.

¬†

`ComplexImageExpression` **`ExprSize`**`( ImageReference` *`image`*`, ComplexImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to the size of image expression and this function will return the value parameter.

¬†

`RealImageExpression` **`ExprSize`**`( ImageReference` *`image`*`, RealImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a real image expression that would otherwise be of an undefined size. The size is set to the size of image expression and this function will return the value parameter.

¬†

`RealImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, RealImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a real image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.

¬†

`RealImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, RealImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a real image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.

¬†

`void` **`Extract2D_Linear`**`( ImageReference` *`from`*`, ImageReference` *`to`*`, Number` *`extract_style`*`, Number` *`x_start`*`, Number` *`y_start`*`, Number` *`x_scale_0`*`, Number` *`y_scale_0`*`, Number` *`x_scale_1`*`, Number` *`y_scale_1`* `)`

¬†

`RealNumberExpression` **`Factorial`**`( Number )`

Return the factorial of the number.

¬†

`ComplexImage` **`FFT`**`( ComplexImage` *`source`* `)`

Creates a new complex 8-byte image from the FFT of the complex image source.

¬†

`ImageReference` **`FindFrontImage`**`( )`

Returns the front image. Doesn't throw exceptions.

¬†

`Function` **`FindFunctionBySignature`**`( String` *`signature`* `)`

Looks for a function that matches the given signature.

¬†

`ImageReference` **`FindImageByID`**`( Number` *`id`* `)`

Returns the image having the given id, or an invalid image of no image has that id.

¬†

`ImageReference` **`FindImageByIndex`**`( Number` *`index`* `)`

Returns the 'index'th image.

¬†

`ImageReference` **`FindImageByLabel`**`( String` *`label`* `)`

Returns the image having the given label, or an invalid image if no such image exists.

¬†

`ImageReference` **`FindImageByName`**`( String` *`name`* `)`

Returns the image having the given name, or an invalid image of no image has that name.

¬†

`ImageReference` **`FindNextImage`**`( ImageReference )`

Find the next image.

¬†
ImageReference **FirstImage**( )
Find the first image.
¬†
void **FlipHorizontal**(ImageReference)
Flips the image horizontally.
¬†
void **FlipVertical**(ImageReference)
Flips the image vertically.
¬†
void **FloatingModelessDialog**(String *prompt*, String *buttonName*, Number *semaphore*)
Present a floating window with the prompt and buttonName. When the user presses the button, the semaphore will be cleared. This function can only be used in the background.
¬†
RealNumberExpression **Floor**(RealNumberExpression)
Return the number truncated to an integer (rounding towards negative infinity).
¬†
void **FM_ConjMultiplyPackedByPacked**(ImageReference *a*, ImageReference *b*)
Conjugate multiply packed image a by packed image b and store the result in b. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ConvertInt16ToFloat**(ImageReference *a*, ImageReference *b*)
Convert the signed 2-byte data in image a to real data and store the result in image b. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ConvertUInt8ToDisplay8**(ImageReference *src*, Number *top*, Number *left*, Number *bottom*, Number *right*, Number *dst*, Number *rowBytes*)
Copy the sub-area of unsigned 1-byte integer image src indicated by [top,left,bottom,right] to dst. The rowBytes parameter indicates the length of the row of dst. To copy to the screen, pass 0 for dst and rowBytes. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ConvertUInt8ToFloat**(ImageReference *a*, ImageReference *b*)
Convert the unsigned 1-byte data in image a to real data and store the result in image b. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_FinishDMA**(ImageReference *image*)
Finish an image for DMA access. This function should be called once for every FM_PrepareDMA() call.
¬†
void **FM_Flush**( )
Finish any pending operations on the array processor.
¬†
Number **FM_GetVarianceFloat**(ImageReference)
Return the variance of the image. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ImageDataChanged**(ImageReference *image*)
Inform the application that the image has changed due to DMA or other memory access.
¬†
void **FM_MultiplyFloatByFloat**(ImageReference *a*, ImageReference *b*)
Multiply real image a by real image b and store the result in a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_MultiplyPackedByFloat**(ImageReference *a*, ImageReference *b*)
Multiply packed image a by real image b and store the result in a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_MultiplyPackedByScalar**(ImageReference *a*, Number)
Multiply packed image a by real and store the result in a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_PackedFFT**(ImageReference *a*)
Perform an in-place packed FFT on image a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_PackedIFFT**(ImageReference)
Perform an in-place packed inverse FFT on image a. No data type checking is performed. This function uses the array processor if present.

¬†

void **FM_PackedLnModulusToImage**(ImageReference *imageSrc*, ImageReference *imageDst*, Number *lowLimit*, Number *highLimit*, Number *range*)

Store the unpacked log modulus of the packed complex image imageSrc into imageDst using the lowLimit, highLimit, and range parameters. No data type checking is performed. This function uses the array processor if present.

¬†

void **FM_PrepareDMA**(ImageReference *image*, NumberVariable *logicalAddress*, NumberVariable *physicalAddress*)

Prepare an image for DMA access by locking down the data and preventing it from being deleted. If DMA access is possible, store the physical address in the physicalAddress variable. Store the logical address in the logicalAddress variable.The FM_FinishDMA() function should be called once for every FM_PrepareDMA() call.

¬†

void **FM_SetMacOnly**(Boolean *b*)

Disable the plug-in array processor if present. This function may not be supported by plug-in array processors.

¬†

void **FM_ShiftCenterFloat**(ImageReference *a*)

Shift the center of image a in-place (after a forward FFT or before an inverse FFT). No data type checking is performed. This function uses the array processor if present.

¬†

void **FM_SubtractMeanPacked**(ImageReference *a*)

Subtract the mean from the packed image a. No data type checking is performed. This function uses the array processor if present.

¬†

void **FM_SubtractMultiply**(ImageReference *a*, ImageReference *b*, ImageReference *c*)

Subtract real image b from real image a, multiply the result by real image c, and store the result in image a. No data type checking is performed. This function uses the array processor if present.

¬†

void **FM_SurveyNormalPackedModulus**(ImageReference *image*, NumberVariable *lowLimit*, NumberVariable *highLimit*)

Survey packed image a and store the minimum and maximum values found into the lowLimit and highLimit variables. No data type checking is performed. This function uses the array processor if present.

¬†

void **FM_TurboFFT**(ImageReference *buffer*, ImageReference *fft*, ImageReference *dark*, ImageReference *gain*, Number *top*, Number *left*, Number *bottom*, Number *right*, Number *dst*, Number *rowBytes*)

Perform a turbo FFT on the real image fft minus the real dark image and multiplied by the real gain image. Store the resulting FFT into the packed complex image fft. Copy the sub-area of unsigned 1-byte integer image src indicated by [top,left,bottom,right] to dst. The rowBytes parameter indicates the length of the row of dst. To copy to the screen, pass 0 for dst and rowBytes. The real buffer image is a scratch area. No data type checking is performed. This function uses the array processor if present.

¬†

Boolean **FontManagerLookupFont**(String *base_name*, Number *text_encoding*, Number *kind*, String *face_name*, NumberVariable *size*)

Tries to match a font to the 'base_name', 'text_encoding', and 'kind' specified.

¬†

String **format**(Number, String *formatString*)

Convert the number to a string using the printf-style real format specified by formatString.

¬†

RealImageExpression **FourPointAverage**(RealImage *source*)

Returns an image expression that evaluates to half the size of the source expression. Each point in the return image expression is the average of the four corresponding points in the source expression.

¬†

ComplexImageExpression **FourPointAverage**(ComplexImage *source*)

Returns an image expression that evaluates to half the size of the source expression. Each point in the return image expression is the average of the four corresponding points in the source expression.

¬†

RGBImageExpression **FourPointAverage**(RGBImage *source*)

Returns an image expression that evaluates to half the size of the source expression. Each point in the return image expression is the average of the four corresponding points in the source expression.

¬†

Number **FPClassify**(Number *v*)

Returns the class of the floating point number.

¬†

void **FreeSemaphore**(Number)

Free the semaphore. Used only with background processing.

¬†

Function **FunctionNullify**(!)

Assigns NULL to dst function.

¬†

`RealNumberExpression` **`Gamma`**`( Number )`

Return the gamma of the number.

¬†

`RealNumberExpression` **`GammaP`**`( Number, Number )`

Return the incomplete gamma function of two numbers.

¬†

`RealNumberExpression` **`GammaQ`**`( Number, Number )`

Return the complement of the incomplete gamma function of two numbers.

¬†

`RealNumberExpression` **`GammaRandom`**`( Number )`

Return a random number with gamma distribution between [0,1).

¬†

`RealNumberExpression` **`GaussianRandom`**`( )`

Return a random number with gaussian distribution between [0,1).

¬†

`void` **`Get1DSize`**`( ImageReference, NumberVariable` *`d0`* `)`

Store the length of the 1D image into the d0 variable.

¬†

`void` **`Get2DSize`**`( ImageReference, NumberVariable` *`d0`*`, NumberVariable` *`d1`* `)`

Store the width and height of the 2D image into the d0 and d1 variables.

¬†

`void` **`Get3DSize`**`( ImageReference, NumberVariable` *`d0`*`, NumberVariable` *`d1`*`, NumberVariable` *`d2`* `)`

Store the x,y, and z sizes of the 3D image into the d0, d1, and d2 variables.

¬†

`Boolean` **`GetAnnotationComplexNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, ComplexNumberVariable` *`noteValue`* `)`

Stores the value of the annotation note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

`Boolean` **`GetAnnotationNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, NumberVariable` *`noteValue`* `)`

Stores the value of the annotation note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

`void` **`GetAnnotationRect`**`( ImageReference, Number` *`annotationID`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`

Store the bounds of the annotation indicated by the annotationID within the image into the top, left, bottom, and right variables.

¬†

`Boolean` **`GetAnnotationRectNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`

Gets the value of the annotation note with label noteLabel into the top, left, bottom, and right variables. Returns 1 if the note exists returns 0 otherwise.

¬†

`Boolean` **`GetAnnotationRGBNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, RGBNumberVariable` *`noteValue`* `)`

Stores the value of the annotation note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

`Boolean` **`GetAnnotationStringNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, String` *`noteValue`* `)`

Copies the value of the annotation note with the label noteLabel into the noteBuffer variable. Returns 1 if the note exists returns 0 otherwise.

¬†

`String` **`GetAnnotationStringNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`* `)`

Returns the value of the annotation note with the label noteLabel. Returns an empty string if the note does not exist.

¬†

`String` **`GetApplicationDirectory`**`( Number` *`index`*`, Boolean` *`create_if_necessary`* `)`

Return one of the application directories. 0=current directory, 1=executable directory.

¬†

`void` **`GetApplicationFont`**`( Number` *`which_font`*`, String` *`face_name`*`, NumberVariable` *`attributes`*`, NumberVariable` *`size`*`, NumberVariable` *`text_encoding`* `)`

Gets the face name, attributes, size, and text encoding of the selected appliction font.

¬†

`Boolean` **`GetApplicationInfo`**`( Number` *`info_kind`*`, NumberVariable` *`info`* `)`

Get application info. For 'info_kind == 0', bit 2 of '*info' is set for demo versions.

¬†

`DocumentWindow` **`GetApplicationWindow`**`( )`

Gets the application window.

¬†

Boolean **GetBoolean**(String *prompt*, Boolean *initalValue*, NumberVariable *result*)

Puts up a dialog with the given prompt and allows the user to enter Boolean. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.

¬†

Boolean **GetCalibrationDialog**(Number *aw*, Number *ah*, NumberVariable *xs*, NumberVariable *ys*, String *initialUnitString*, String *unitString*)

Present the calibration dialog to the user. The calibrating pixel dimensions are passed as the [aw,ah] parameters. The resulting calibration is stored into the [xs,ys] parameters. The initial unit string is passed in and the resulting unit string is stored into the unitString variable.Returns 1 for OK and 0 for Cancel.

¬†

RGBImage **GetCLUT**(ImageReference)

Return the image's CLUT as a 256x1 RGB image.

¬†

Boolean **GetComplexNumberNote**(ImageReference, String *noteLabel*, ComplexNumberVariable *noteValue*)

Stores the value of the image note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

String **GetDate**(Number *dateFormat*)

Return a string representing the current date in the date format indicated by dateFormat. The dateFormat parameter can be 0=short, 1=long, 2=abbreviated.

¬†

Boolean **GetDirectoryDialog**(String *dirName*)

Puts up the GetDirectory dialog and stores the path of the chosen directory in 'dirName'

¬†

RGBImage **GetDisplayAsImage**(ImageReference)

Convert the display of image to an RGB image.

¬†

DocumentWindow **GetDocumentWindow**(Number *index*)

Gets the 'index'th document window.

¬†

DocumentWindow **GetDocumentWindowByTitle**(String *name*)

Gets the document window named 'name'.

¬†

void **GetEstimatedMinMax**(ImageReference, NumberVariable *minPtr*, NumberVariable *maxPtr*)

Store the current estimated minimum and maximum of the image into the minPtr and maxPtr variables.

¬†

String **GetExceptionDescription**( )

Return the message that would be displayed in the error dialog box for an exception as a string.

¬†

String **GetExceptionString**( )

Return the message that would be displayed in the error dialog box for an exception as a string.

¬†

TagGroup **GetFilesInDirectory**(String *path*, Number *search_flags*)

Returns a tag group containing a list of the file names in the directory 'dir_path'

¬†

Number **GetFileSize**(Number *file*)

Returns the size of the file.

¬†

DocumentWindow **GetFloatingWindow**(Number *index*)

Gets the 'index'th floating window.

¬†

Boolean **GetFourImages**(String *title*, ImageVariable *image1*, ImageVariable *image2*, ImageVariable *image3*, ImageVariable *image4*)

Puts up a dialog and allows the user to choose four images. Returns 1 for Ok and 0 for Cancel.

¬†

Boolean **GetFourImagesWithPrompt**(String *prompt*, String *title*, ImageVariable *image1*, ImageVariable *image2*, ImageVariable *image3*, ImageVariable *image4*)

Puts up a dialog with the given prompt and allows the user to choose four images. Returns 1 for Ok and 0 for Cancel.

¬†

Boolean **GetFourLabeledImagesWithPrompt**(String *prompt*, String *title*, String *label1*, ImageVariable *image1*, String *label2*, ImageVariable *image2*, String *label3*, ImageVariable *image3*, String *label4*, ImageVariable *image4*)

Puts up a dialog with the given prompt and allows the user to choose four images. Returns 1 for Ok and 0 for Cancel.

¬†

Boolean **GetFrontImage**(ImageVariable)

Set the image variable to represent the foremost image, returns 1 if successful; return 0 otherwise.
¬†

`ImageReference` **`GetFrontImage`**`( )`
Return the foremost image.
¬†

`ImageDocument` **`GetFrontImageDocument`**`( )`
Returns the front image document.
¬†

`Number` **`GetFrontImageID`**`( )`
Return the id of the front most image window.
¬†

`Number` **`GetImageDataSeed`**`( ImageReference )`
Return a seed representing the data of the image. Each time the image data changes, the seed will change.
¬†

`ImageDocument` **`GetImageDocument`**`( Number` *`position`* `)`
Returns the image document by position with the application.
¬†

`ImageDocument` **`GetImageDocumentByID`**`( Number` *`id`* `)`
Returns the image document whose id is 'id'.
¬†

`ImageReference` **`GetImageFromID`**`( Number` *`imageID`* `)`
Return the image corresponding the imageID.
¬†

`Boolean` **`GetImageFromID`**`( ImageVariable, Number` *`imageID`* `)`
Store the image corresponding the imageID into the image variable. Return 1 if one is found; return 0 otherwise.
¬†

`Number` **`GetImageID`**`( ImageReference )`
Return the id of the image.
¬†

`Boolean` **`GetInteger`**`( String` *`prompt`*`, Number` *`initalValue`*`, NumberVariable` *`result`* `)`
Puts up a dialog with the given prompt and allows the user to enter an integer. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.
¬†

`Boolean` **`GetInversionMode`**`( ImageReference )`
Return the contrast inversion mode of the image (1=inverted, 0=not inverted).
¬†

`Number` **`GetKey`**`( )`
Returns the key that was last pressed.
¬†

`String` **`GetLabel`**`( ImageReference )`
Return the image label of the image.
¬†

`void` **`GetLimits`**`( ImageReference, NumberVariable` *`lowPtr`*`, NumberVariable` *`highPtr`* `)`
Stores display limits into the lowPtr and highPtr variables.
¬†

`void` **`GetMaximalDocumentWindowRect`**`( Number` *`options`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the bounds of the content region of the largest document window.
¬†

`String` **`GetName`**`( ImageReference )`
Return the name of the image's image document.
¬†

`ImageReference` **`GetNamedImage`**`( String` *`name`* `)`
Return the image with the image document name.
¬†

`Boolean` **`GetNamedImage`**`( ImageVariable, String` *`name`* `)`
Store the image with the image document name into the image variable. Return 1 if one is found; return 0 otherwise.
¬†

`Number` **`GetNextImageID`**`( Number` *`id`* `)`
Return the id of the image window following the image with the given id.
¬†

`Boolean` **`GetNoteState`**`( ImageReference, String` *`noteLabel`*`, NumberVariable` *`hidden`* `)`
Stores whether the image note with the label noteLabel is in the copy-to-image list into the copyToImage parameter. The hidden parameter is no longer used.

¬†

Number **GetNthAnnotationID**(ImageReference, Number *index*)

Return the ID of the index'th annotation in the image.

¬†

DocumentWindow **GetNthDocumentWindowOfType**(Number *type*, Number *index*)

Returns the 'index'th document window of type 'type'.

¬†

Number **GetNthImageID**(Number *n*)

Return the id of the nth image (number from 0). The images are in no particular order.

¬†

Boolean **GetNthPersistentNoteTitle**(String *tagPath*, Number *index*, StringVariable *noteValue*)

Stores the title of the tag with the given index from the persistent tag group indicated by tagPath into noteValue.

¬†

String **GetNthPersistentNoteTitle**(String *tagPath*, Number *index*)

Returns the title of the tag with the given index from the persistent tag group indicated by tagPath.

¬†

void **GetNthPersistentNumberNote**(String *tagPath*, Number *index*, NumberVariable *noteValue*)

Stores the number with the given index from the persistent tag list indicated by tagPath into noteValue.

¬†

Boolean **GetNumber**(String *prompt*, Number *initalValue*, NumberVariable *result*)

Puts up a dialog with the given prompt and allows the user to enter a number. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.

¬†

Boolean **GetNumberNote**(ImageReference, String *noteLabel*, NumberVariable *noteValue*)

Stores the value of the image note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

Boolean **GetOneImage**(String *title*, ImageVariable *image1*)

Puts up a dialog and allows the user to choose an image. Returns 1 for Ok and 0 for Cancel.

¬†

Boolean **GetOneImageWithPrompt**(String *prompt*, String *title*, ImageVariable *image1*)

Puts up a dialog and allows the user to choose an image. Returns 1 for Ok and 0 for Cancel.

¬†

Boolean **GetOneLabeledImageWithPrompt**(String *prompt*, String *title*, String *label1*, ImageVariable *image1*)

Puts up a dialog and allows the user to choose an image. Returns 1 for Ok and 0 for Cancel.

¬†

void **GetOrigin**(ImageReference, NumberVariable *x*, NumberVariable *y*)

Store the origin of image into the x and y variables. The origin is in the same units as scale.

¬†

Number **GetOSTickCount**( )

Return a tick count appropriate for the operating system.

¬†

Number **GetOSTicksPerSecond**( )

Return the number of ticks per second of a tick count appropriate for the operating system.

¬†

TagGroup **GetPackageTags**(String *identifier*)

Return the tags specified by identifier. The identifier is used to identify tags loaded with a specific package.

¬†

Boolean **GetPersistentComplexNumberNote**(String *noteLabel*, ComplexNumberVariable *noteValue*)

Stores the value of the persistent note with label noteLabel into the noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

Boolean **GetPersistentLongNote**(String *noteLabel*, NumberVariable *noteValue*)

Stores the value of the persistent note with label noteLabel into the noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

Boolean **GetPersistentNoteState**(String *noteLabel*, NumberVariable *copyToImage*, NumberVariable *hidden*)

Stores whether the note with the label noteLabel is in the copy-to-image list into the copyToImage parameter. The hidden parameter is no longer used.

¬†

Boolean **GetPersistentNumberNote**(String *noteLabel*, NumberVariable *noteValue*)

Stores the value of the persistent note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.

¬†

Boolean **GetPersistentPointNote**(String *noteLabel*, NumberVariable *x*, NumberVariable *y*)

Gets the value of the persistent note with label noteLabel into the x and y variables. Returns 1 if the note exists returns 0 otherwise.

¬†

Boolean **GetPersistentRectNote**(String *noteLabel*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the value of the persistent note with label noteLabel into the top, left, bottom, and right variables. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentRGBNumberNote**(String *noteLabel*, RGBNumberVariable *noteValue*)
Stores the value of the persistent note with label noteLabel into the noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentStringNote**(String *noteLabel*, String *noteBuffer*)
Copies the value of the persistent note with the label noteLabel into the noteBuffer variable. Returns 1 if the note exists returns 0 otherwise.
¬†

String **GetPersistentStringNote**(String *noteLabel*)
Returns the value of the persistent note with the label noteLabel. Returns an empty string if the note does not exist.
¬†

TagGroup **GetPersistentTagGroup**( )
Gets the persistent tag group.
¬†

RGBNumber **GetPixel**(RGBImage, Number *x*, Number *y*)
Return the pixel in the image at [x,y].
¬†

ComplexNumber **GetPixel**(ComplexImage, Number *x*, Number *y*)
Return the pixel in the image at [x,y]. Does not work on packed complex images.
¬†

Number **GetPixel**(RealImage, Number *x*, Number *y*)
Return the pixel in the image at [x,y].
¬†

Number **GetPlatformInfo**(Number *info*)
Return platform info. info=1 is general platform (1=MacOS,2=Windows).
¬†

Boolean **GetPointNote**(ImageReference, String *noteLabel*, NumberVariable *x*, NumberVariable *y*)
Gets the value of the image note with label noteLabel into the x and y variables. Returns 1 if the note exists returns 0 otherwise.
¬†

Number **GetProcessID**( )
Returns the current process id.
¬†

void **GetRawStreamPos**(Number *rawStream*, NumberVariable *pos*)
Store the current position in rawStream into the pos variable.
¬†

void **GetRawStreamSize**(Number *rawStream*, NumberVariable *size*)
Store the length of rawStream into the size variable.
¬†

Boolean **GetRectNote**(ImageReference, String *noteLabel*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the value of the image note with label noteLabel into the top, left, bottom, and right variables. Returns 1 if the note exists returns 0 otherwise.
¬†

DocumentWindow **GetResultsWindow**(Boolean *open*)
Gets the results window. If the window is not open, and 'open' is true, the results window is opened.
¬†

Boolean **GetRGBColorDialog**(String *prompt*, RGBNumber *defaultColor*, RGBNumberVariable *result*)
Puts up the Color Picker dialog with the given prompt and allows the user to choose a color. The default color is passed in and the resulting color is stored in the result parameter. Returns 1 for OK and 0 for Cancel.
¬†

Boolean **GetRGBNumberNote**(ImageReference, String *noteLabel*, RGBNumberVariable *noteValue*)
Stores the value of the image note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

ROI **GetROIFromID**(Number *id*)
Returns the region of interest associated with the ID or NULL if it does not exist.
¬†

void **GetScale**(ImageReference, NumberVariable *x*, NumberVariable *y*)
Store the scale of image into the x and y variables.
¬†

void **GetScreenSize**(NumberVariable *width*, NumberVariable *height*)
Store the size of the screen into the width and height variables.

¬†

`ScriptObject` **`GetScriptObjectFromID`**`( Number` *`id`* `)`

Returns the script object associated with the ID or NULL if the object does not exist.

¬†

`Boolean` **`GetSelection`**`( ImageReference, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`

Stores the coordinates (in pixels) of the image's selection into the top, left, bottom, and right variables. Returns 1 if there was a selection and 0 if there wasn't.

¬†

`void` **`GetSize`**`( ImageReference, NumberVariable` *`width`*`, NumberVariable` *`height`* `)`

Store the width and height of the 2D image into the width and height variables.

¬†

`String` **`GetSpecialDirectory`**`( Number` *`index`* `)`

Return one of the special directories. 0=current directory, 1=executable directory.

¬†

`Number` **`GetSpecialWindow`**`( Number` *`index`* `)`

Return one of the special windows. On Windows, 0=frame window, 1=top-most dialog.

¬†

`Boolean` **`GetString`**`( String` *`prompt`*`, String` *`initalValue`*`, String` *`result`* `)`

Puts up a dialog with the given prompt and allows the user to enter a string. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.

¬†

`Boolean` **`GetStringFromList`**`( ImageReference, String` *`tagPath`*`, Number` *`index`*`, String` *`noteBuffer`* `)`

Copies the string with the given index from the image tag list indicated by tagPath to noteBuffer.

¬†

`String` **`GetStringFromList`**`( ImageReference, String` *`tagPath`*`, Number` *`index`* `)`

Returns the string with the given index from the image tag list indicated by tagPath.

¬†

`String` **`GetStringFromPersistentList`**`( String` *`tagPath`*`, Number` *`index`* `)`

Returns the string with the given index from the persistent tag list indicated by tagPath.

¬†

`Boolean` **`GetStringFromPersistentList`**`( String` *`tagPath`*`, Number` *`index`*`, String` *`noteBuffer`* `)`

Copies the string with the given index from the persistent tag list indicated by tagPath to noteBuffer.

¬†

`String` **`GetStringNote`**`( ImageReference, String` *`noteLabel`* `)`

Returns the value of the image note with the label noteLabel. Returns an empty string if the note does not exist.

¬†

`Boolean` **`GetStringNote`**`( ImageReference, String` *`noteLabel`*`, String` *`noteValue`* `)`

Copies the value of the image note with the label noteLabel into the noteBuffer variable. Returns 1 if the note exists returns 0 otherwise.

¬†

`Boolean` **`GetThreeImages`**`( String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`*`, ImageVariable` *`image3`* `)`

Puts up a dialog and allows the user to choose three images. Returns 1 for Ok and 0 for Cancel.

¬†

`Boolean` **`GetThreeImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`*`, ImageVariable` *`image3`* `)`

Puts up a dialog with the given prompt and allows the user to choose three images. Returns 1 for Ok and 0 for Cancel.

¬†

`Boolean` **`GetThreeLabeledImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, String` *`label1`*`, ImageVariable` *`image1`*`, String` *`label2`*`, ImageVariable` *`image2`*`, String` *`label3`*`, ImageVariable` *`image3`* `)`

Puts up a dialog with the given prompt and allows the user to choose three images. Returns 1 for Ok and 0 for Cancel.

¬†

`Number` **`GetTicks`**`( )`

Return the MacOS system tick count.

¬†

`Number` **`GetTicksPerSecond`**`( )`

Return the number of ticks per second.

¬†

`String` **`GetTime`**`( Boolean` *`wantSeconds`* `)`

Return a string representing the current time with or without seconds as indicated by the wantSeconds parameter.

¬†

`Boolean` **`GetTwoImages`**`( String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`* `)`

Puts up a dialog and allows the user to choose two images. Returns 1 for Ok and 0 for Cancel.

¬†

`Boolean` **`GetTwoImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable`

image2)
Puts up a dialog with the given prompt and allows the user to choose two images. Returns 1 for Ok and 0 for Cancel.
¬†
Boolean **GetTwoLabeledImagesWithPrompt**(String *prompt*, String *title*, String *label1*, ImageVariable *image1*, String *label2*, ImageVariable *image2*)
Puts up a dialog with the given prompt and allows the user to choose two images. Returns 1 for Ok and 0 for Cancel.
¬†
Number **GetUnitsH**(ImageReference, Number *x*)
Return the horizontal pixels x in calibrated units.
¬†
String **GetUnitString**(ImageReference)
Returns the unit string of the image.
¬†
Number **GetUnitsV**(ImageReference, Number *y*)
Return the vertical pixels y in calibrated units.
¬†
void **GetWindowPosition**(ImageReference, NumberVariable *xPos*, NumberVariable *yPos*)
Store the image's image document window position into the xPos and yPos variables.
¬†
void **GetWindowSize**(ImageReference, NumberVariable *width*, NumberVariable *height*)
Store the image's image document window size into the width and height variables.
¬†
Number **GetZoom**(ImageReference)
Return the zoom of the image display.
¬†
void **GrabSemaphore**(Number)
Grab the semaphore. Block until it is available. Used only with background processing.
¬†
RealNumberExpression **green**(RGBNumberExpression)
Return the green portion of an RGB number.
¬†
void **GroupAnnotationUngroup**(Component *comp*)
Ungroups the group annotation.
¬†
Boolean **HasAcquisitionDaemon**(ImageReference)
Returns 1 if the image has an attached daemon and returns 0 otherwise.
¬†
String **Hex**(Number *n*)
Returns the number as a hexidecimal string.
¬†
String **Hex**(Number *n*, Number *length*)
Returns the number as a hexidecimal string of the given length.
¬†
void **HideImage**(ImageReference)
Hide the image's image document.
¬†
ComplexImage **IFFT**(ComplexImage *source*)
Creates a new complex 8-byte image from the inverse FFT of the complex image source.
¬†
void **ImageData_CacheChanged**(Number *imageDataID*)
Tell the array processor manager that the cached image has changed. See the SDK documentation for more information.
¬†
void **ImageData_Changed**(Number *imageDataID*)
Tell the array processor manager that the image has changed. See the SDK documentation for more information.
¬†
void **ImageData_FlushCache**(Number *imageDataID*)
Tell the array processor manager to flush the image from the cache. See the SDK documentation for more information.
¬†
Number **ImageData_GetSeed**(Number *imageDataID*)
Ask the array processor manager for the data seed of the image. See the SDK documentation for more information.
¬†
Boolean **ImageData_IsLocalCopyValid**(Number *imageDataID*)
Ask the array processor manager if the local copy is valid or not. See the SDK documentation for more information.
¬†
void **ImageData_SetLocalSeed**(Number *imageDataID*, Number *value*)

Tell the array processor manager to set the local data seed. See the SDK documentation for more information.
¬†
ImageDisplay **ImageDisplayNullify**(!)
¬†
ImageDocument **ImageDocumentNullify**(!)
¬†
ImageReference **ImageNullify**(!)
¬†
ImageDocument **ImageWindowGetImageDocument**(DocumentWindow *window*)
Gets the image document displayed in the window.
¬†
RealNumberExpression **imaginary**(ComplexNumberExpression)
Return the imaginary portion of the complex number.
¬†
RealImageExpression **Index**(RealImage, RealImageExpression *x*, RealImageExpression *y*)
Returns a pixel in the given real image at the position [x,y]. Performs bounds checking.
¬†
RealImageExpression **Index**(RealImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)
Returns a pixel in the given real image at the position [x,y,z]. Performs bounds checking.
¬†
ComplexImageExpression **Index**(ComplexImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)
Returns a pixel in the given Complex image at the position [x,y,z]. Performs bounds checking.
¬†
RGBImageExpression **Index**(RGBImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)
Returns a pixel in the given RGB image at the position [x,y,z]. Performs bounds checking.
¬†
ComplexImageExpression **Index**(ComplexImage, RealImageExpression *x*, RealImageExpression *y*)
Returns a pixel in the given complex image at the position [x,y]. Performs bounds checking.
¬†
RGBImageExpression **Index**(RGBImage, RealImageExpression *x*, RealImageExpression *y*)
Returns a pixel in the given RGB image at the position [x,y]. Performs bounds checking.
¬†
Number **InstallScriptLibraryFile**(String *fileName*)
Loads the script file indicated by fileName, executes it, and publishes any functions contained inside. Always returns 0.
¬†
RealImage **IntegerImage**(String *title*, Number *bytes*, Boolean *isSigned*, Number *d0*)
Creates a 1D integer image of size [d0] with the given title. The bytes and isSigned parameters specify integer specific attributes of the data.
¬†
RealImage **IntegerImage**(String *title*, Number *bytes*, Boolean *isSigned*, Number *d0*, Number *d1*, Number *d2*)
Creates a 3D integer image of size [d0,d1,d2] with the given title. The bytes can be 1, 2, or 4 and isSigned can be 1 (true) or 0 (false).
¬†
RealImage **IntegerImage**(String *title*, Number *bytes*, Boolean *isSigned*, Number *d0*, Number *d1*)
Creates a 2D integer image of size [d0,d1] with the given title. The bytes and isSigned parameters specify integer specific attributes of the data.
¬†
Number **integrate**(RealImageExpression)
Return the sum of all pixel values of the image expression.
¬†
Boolean **IsAnnotationSelected**(ImageReference, Number *annotationID*)
Return 1 if the annotation indicated by the annotationID within the image is selected; returns 0 otherwise.
¬†
Boolean **IsBinaryDataType**(ImageReference)
Returns 1 if the image is an binary data type; returns 0 otherwise.
¬†
Boolean **IsByteImage**(ImageReference)
Returns 1 if the image is unsigned 1-byte integer data; returns 0 otherwise.
¬†
Boolean **IsComplexDataType**(ImageReference, Number *bytes*)
Returns 1 if the image is an complex data type of size bytes; returns 0 otherwise.
¬†

Boolean **IsComplexImage**(ImageReference)
Returns 1 if the image is single precision complex data; returns 0 otherwise.
¬†
Boolean **IsDisplayValid**(ImageReference)
Return 1 if the image's display is up-to-date and 0 otherwise.
¬†
Boolean **IsExceptionUserAbort**( )
Returns true if the exception currently in effect is a user abort.
¬†
Boolean **IsFloatImage**(ImageReference)
Returns 1 if the image is single precision real data; returns 0 otherwise.
¬†
Boolean **IsImageComplex**(Number *id*)
Return true or false to indicate whether given image with the given id is complex-valued or not.
¬†
Boolean **IsImageReal**(Number *id*)
Return true or false to indicate whether given image with the given id is real-valued or not.
¬†
Boolean **IsImageRGB**(Number *id*)
Return true or false to indicate whether given image with the given id is RGB-valued or not.
¬†
Boolean **IsInfinite**(Number *v*)
Returns true if the value is infinite.
¬†
Boolean **IsIntegerDataType**(ImageReference, Number *bytes*, Boolean *isSigned*)
Returns 1 if the image is an integer data type of size bytes with a matching sign charateristic as signed; returns 0 otherwise.
¬†
Boolean **IsLongImage**(ImageReference)
Returns 1 if the image is signed 4-byte integer data; returns 0 otherwise.
¬†
Boolean **IsNan**(Number *v*)
Returns true if the value is not a number.
¬†
Boolean **IsPackedComplexImage**(ImageReference)
Returns 1 if the image is packed complex data; returns 0 otherwise.
¬†
Boolean **IsRealDataType**(ImageReference, Number *bytes*)
Returns 1 if the image is an real data type of size bytes; returns 0 otherwise.
¬†
Boolean **IsRGBDataType**(ImageReference, Number *bytes*)
Returns 1 if the image is an RGB data type of size bytes; returns 0 otherwise.
¬†
Boolean **IsShortImage**(ImageReference)
Returns 1 if the image is signed 2-byte integer data; returns 0 otherwise.
¬†
void **KeepImage**(ImageReference)
Keep the image from being deleted automatically when the image's script scope is exited.
¬†
void **KillProcess**(Number *pid*)
Kills the process specified by pid.
¬†
String **left**(String, Number *count*)
Returns leftmost count characters of a string.
¬†
RealNumberExpression **LegendrePolynomial**(Number, Number, Number)
Return the Legendre polynomial function for a number.
¬†
Number **len**(String)
Returns the length of a string.
¬†
LinePlotImageDisplay **LinePlotImageDisplayNullify**(!)
¬†
void **LoadNotes**(ImageReference, String *tagPath*, String *fileName*)
Loads the image notes from the given fileName into tagPath. This function will read the group name from the file. The tagPath must refer

to a group.
¬†
`void` **`LoadNotesAs`**`( ImageReference, String` *`tagPath`*`, String` *`fileName`*` )`
Loads the image notes from the given fileName into tagPath. The tagPath must refer to a group.
¬†
`void` **`LoadPersistentNotes`**`( String` *`tagPath`*`, String` *`fileName`*` )`
Loads the persistent notes from the given fileName into tagPath. This function will read the group name from the file. The tagPath must refer to a group.
¬†
`void` **`LoadPersistentNotesAs`**`( String` *`tagPath`*`, String` *`fileName`*` )`
Loads the persistent notes from the given fileName into tagPath. The tagPath must refer to a group.
¬†
`ComplexNumberExpression` **`log`**`( ComplexNumberExpression )`
Return the logarithm of the number.
¬†
`RealNumberExpression` **`log`**`( RealNumberExpression )`
Return the natural logarithm of the number.
¬†
`RealNumberExpression` **`log1`**`( RealNumberExpression )`
Return the natural logarithm of the number + 1.
¬†
`RealNumberExpression` **`log10`**`( RealNumberExpression )`
Return the logarithm base 10 of the number.
¬†
`RealNumberExpression` **`log2`**`( RealNumberExpression )`
Return the logarithm base 2 of the number.
¬†
`RealNumberExpression` **`LogGamma`**`( Number )`
Return the log gamma of the number.
¬†
`RealImage` **`LUDecomposition`**`( RealImage` *`a`*`, RealImage` *`b`*` )`
Return the image resulting from a LU decomposition on images a,b.
¬†
`Number` **`MatrixDeterminant`**`( RealImage` *`a`*` )`
Return the matrix determinant number of matrix image a.
¬†
`RealImage` **`MatrixInverse`**`( RealImage` *`a`*` )`
Return the matrix inverse image of matrix image a.
¬†
`RealImage` **`MatrixMultiply`**`( RealImage` *`a`*`, RealImage` *`b`*` )`
Return the matrix product image of matrix images a and b.
¬†
`void` **`MatrixPrint`**`( ImageReference` *`a`*` )`
Print an image as a matrix to the results window.
¬†
`ComplexImage` **`MatrixTranspose`**`( ComplexImage` *`a`*` )`
Return the matrix transpose image of matrix image a.
¬†
`RGBImage` **`MatrixTranspose`**`( RGBImage` *`a`*` )`
Return the matrix transpose image of matrix image a.
¬†
`RealImage` **`MatrixTranspose`**`( RealImage` *`a`*` )`
Return the matrix transpose image of matrix image a.
¬†
`RealNumberExpression` **`max`**`( RealNumberExpression, RealNumberExpression )`
Return the maximum of the two numbers.
¬†
`Number` **`max`**`( RealImageExpression, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Finds the maximum pixel in the image, stores the coordinates of that pixel into x and y, and returns the value of the pixel.
¬†
`Number` **`max`**`( RealImageExpression )`
Return the maxmimum pixel within the image expression.
¬†
`RealNumberExpression` **`Maximum`**`( RealNumberExpression... )`

Return the maximum of a list of numbers.
¬†
`Number` **`mean`**`( RealImageExpression )`
Return the mean pixel value of the image expression.
¬†
`Number` **`MeanSquare`**`( RealImageExpression )`
Return the mean square of all pixel values of the image expression.
¬†
`RealNumberExpression` **`Median`**`( RealNumberExpression... )`
Return the median of a list of numbers.
¬†
`RealImage` **`MedianFilter`**`( ImageReference` *`source`*`, Number` *`filterType`*`, Number` *`size`*` )`
Performs a median filter on the source image according to the filterType parameter (0=horizontal, 1=vertical, 2=cross, 3=entire) and the size parameter. Size specifies size in each direction - so a 'size' of 2 is a 5x5 window.
¬†
`String` **`mid`**`( String, Number` *`offset`*`, Number` *`count`*` )`
Returns count characters of a string starting at offset.
¬†
`RealNumberExpression` **`min`**`( RealNumberExpression, RealNumberExpression )`
Return the minimum of the two numbers.
¬†
`Number` **`min`**`( RealImageExpression, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Finds the minimum pixel in the image, stores the coordinates of that pixel into x and y, and returns the value of the pixel.
¬†
`Number` **`min`**`( RealImageExpression )`
Return the minimum pixel within the image expression.
¬†
`RealNumberExpression` **`Minimum`**`( RealNumberExpression... )`
Return the minimum of a list of numbers.
¬†
`void` **`minmax`**`( RealImageExpression, NumberVariable` *`minP`*`, NumberVariable` *`maxP`*` )`
Finds the minimum and maximum pixel in the image and store those values into minP and maxP.
¬†
`RealNumberExpression` **`mod`**`( RealNumberExpression, RealNumberExpression )`
Return the modulus of the first number divided by the second number.
¬†
`void` **`ModelessDialog`**`( String` *`prompt`*`, String` *`buttonName`*`, Number` *`semaphore`*` )`
Present a modeless dialog with the prompt and buttonName. When the user presses the button, the semaphore will be cleared. This function can only be used in the background.
¬†
`RealNumberExpression` **`modulus`**`( ComplexNumberExpression )`
Return the modulus of a complex number.
¬†
`void` **`MoveAnnotation`**`( ImageReference, Number` *`annotationID`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Set the bounds of the annotation indicated by the annotationID within the image to [top, left, bottom, right].
¬†
`RealImage` **`MPClose`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically close the image using the neighbors parameter to control the closing and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPDilate`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically dilate the image using the neighbors parameter to control the dilation and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPDistanceMap`**`( ImageReference` *`image`*` )`
Generate a distance map from the source image and return the resulting real image. The source image must be binary.
¬†
`RealImage` **`MPErode`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically erode the image using the neighbors parameter to control the erosion and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPEuclideanDistanceMap`**`( ImageReference` *`image`*` )`
Generate a Euclidean distance map from the source image and return the resulting real image. The source image must be binary.

¬†
ComplexImage **MPExactDistanceMap**(ImageReference *image*)
Generate an exact distance map from the source image and return the resulting complex image. The source image must be binary.
¬†
RealImage **MPOpen**(RealImage *image*, Number *neighbors*)
Morphologically open the image using the neighbors parameter to control the opening and return the resulting image. The source image must be binary.
¬†
RealImage **MPOutline**(RealImage *image*)
Morphologically outline the image and return the resulting image. The source image must be binary.
¬†
Number **Nan**(String *tag*, Boolean *is_signaling*)
Returns a nan containing the specified tag.
¬†
RealNumberExpression **Nearest**(RealNumberExpression)
Return the number rounded to the nearest integer.
¬†
Function **NewAbstractMethod**(String *method_name*, String *method_signature*)
Returns an abstract method with name 'method_name' and signature 'method_signature'.
¬†
Component **NewArrowAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new arrow annotation.
¬†
Component **NewBoxAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new box annotation.
¬†
Function **NewCallbackFunction**(String *method_signature*, ! *callback*, Number *linkage_style*)
Creates a function representing a callback to a C function 'callback'.
¬†
Component **NewComponent**(Number *type*, Number *f1*, Number *f2*, Number *f3*, Number *f4*)
Creates a new annotaiton of type 'type'
¬†
Component **NewDoubleArrowAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new double arrow annotation.
¬†
Function **NewFunctionFromScript**(String *script*, String *signature*)
Compiles script and returns the function matching signature.
¬†
Component **NewGroupAnnotation**( )
Creates a new group annotation.
¬†
ImageReference **NewImage**(String *title*, Number *type*, Number *d0*, Number *d1*, Number *d2*, Number *d3*)
Creates a 4D image of type 'type' and size [d0,d1,d2,d3] with the given title.
¬†
ImageReference **NewImage**(String *title*, Number *type*, Number *d0*, Number *d1*, Number *d2*)
Creates a 3D image of type 'type' and size [d0,d1,d2] with the given title.
¬†
ImageReference **NewImage**(String *title*, Number *type*, Number *d0*, Number *d1*)
Creates a 2D image of type 'type' and size [d0,d1] with the given title.
¬†
ImageReference **NewImage**(String *title*, Number *type*, Number *d0*)
Creates a 1D image of type 'type' and size [d0] with the given title.
¬†
ImageDocument **NewImageDocument**(String *title*)
Creates an empty image document.
¬†
ImageDocument **NewImageDocumentFromFile**(String *path_name*)
Creates a new image document from a file.
¬†
ImageReference **NewImageFromFile**(String *file_path*)
Opens a file and reads it as an image.
¬†
Component **NewLineAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new line annotation.

¬†
ImageReference **NewLiveFFT**(ImageDisplay *imageDisplay*, ROI *roi*, Boolean *reduce*)
Creates a new live fft of the area in 'roi', that is reduced if 'reduce' is 'true'
¬†
ImageReference **NewLiveHistogram**(ImageDisplay *imageDisplay*, ROI *roi*, Number *num_channels*)
Creates a new live histogram of the area in 'roi', binned by 'num_channels'
¬†
ImageReference **NewLiveProfile**(ImageDisplay *imageDisplay*, Number *start_x*, Number *start_y*, Number *end_x*, Number *end_y*, Number *width*)
Creates a new live profile from (start_x,start_y) to (end_x,end_y)
¬†
Component **NewOvalAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new oval annotation.
¬†
Component **NewPictureAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*, Number *picture*)
Creates a new picture annotation.
¬†
ROI **NewROI**( )
Creates an empty region of interest.
¬†
DocumentWindow **NewScriptWindow**(String *title*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new editor window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*, String *font_name*, Number *attributes*, Number *size*, Number *encoding*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Opens a file into a script window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*)
Opens a file into a script window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Opens a file into a script window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*, String *font_name*, Number *attributes*, Number *size*, Number *encoding*)
Opens a file into a script window.
¬†
Number **NewSemaphore**( )
Create a semaphore. Used only with background processing.
¬†
ScriptObject **NewStreamFromBuffer**(Number *initial_size*)
Creates a new stream object from from a memory buffer with initial size 'initial_size'.
¬†
ScriptObject **NewStreamFromFileReference**(Number *file_ref*, Boolean *do_close*)
Creates a new stream object from the file reference. On destruction, closes the file reference if 'do_close' is true.
¬†
TagGroup **NewTagGroup**( )
Creates an empty tag group.
¬†
TagGroup **NewTagList**( )
Creates an empty tag list.
¬†
Component **NewTextAnnotation**(Number *left*, Number *top*, String *text*, Number *size*)
Creates a new text annotation.
¬†
ImageReference **NextImage**(ImageReference)
Find the next image.
¬†
RGBImageExpression **NoClipIndex**(RGBImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)
Returns a pixel in the given RGB image at the position [x,y,z]. Does not perform bounds checking. This is slightly faster than Index().
¬†
RealImageExpression **NoClipIndex**(RealImage, RealImageExpression *x*, RealImageExpression *y*)

Returns a pixel in the given real image at the position [x,y]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`RGBImageExpression` **`NoClipIndex`**`( RGBImage, RealImageExpression x, RealImageExpression y )`
Returns a pixel in the given RGB image at the position [x,y]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`ComplexImageExpression` **`NoClipIndex`**`( ComplexImage, RealImageExpression x, RealImageExpression y, RealImageExpression z )`
Returns a pixel in the given Complex image at the position [x,y,z]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`RealImageExpression` **`NoClipIndex`**`( RealImage, RealImageExpression x, RealImageExpression y, RealImageExpression z )`
Returns a pixel in the given real image at the position [x,y,z]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`ComplexImageExpression` **`NoClipIndex`**`( ComplexImage, RealImageExpression x, RealImageExpression y )`
Returns a pixel in the given Complex image at the position [x,y]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`RealNumberExpression` **`norm`**`( ComplexNumberExpression )`
Return the norm of a complex number. This is the modulus squared.
¬†

`void` **`ObjectTransformCompose`**`( Number i1_o_x, Number i1_o_y, Number i1_s_x, Number i1_s_y, Number i2_o_x, Number i2_o_y, Number i2_s_x, Number i2_s_y, NumberVariable o_o_x, NumberVariable o_o_y, NumberVariable o_s_x, NumberVariable o_s_y )`
Composes the transform (i1_o_x,i1_o_y,i1_s_x,i1_s_y) with (i2_o_x,i2_o_y,i2_s_x,i2_s_y) and places the result in (*o_o_x,*o_o_y,*o_s_x,*o_s_y).
¬†

`void` **`ObjectTransformInvert`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, NumberVariable o_o_x, NumberVariable o_o_y, NumberVariable o_s_x, NumberVariable o_s_y )`
Inverts the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_o_x,*o_o_y,*o_s_x,*o_s_y).
¬†

`void` **`ObjectTransformTransformPoint`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_p_x, Number i_p_y, NumberVariable o_p_x, NumberVariable o_p_y )`
Transforms the point (i_p_x,i_p_y) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_p_x,*o_p_y).
¬†

`void` **`ObjectTransformTransformRect`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_r_t, Number i_r_l, Number i_r_b, Number i_r_r, NumberVariable o_r_t, NumberVariable o_r_l, NumberVariable o_r_b, NumberVariable o_r_r )`
Transforms the point (i_r_t,i_r_l,i_r_b,i_r_r) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_r_t,*o_r_l,*o_r_b,*o_r_r).
¬†

`void` **`ObjectTransformUntransformPoint`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_p_x, Number i_p_y, NumberVariable o_p_x, NumberVariable o_p_y )`
Transforms the point (i_p_x,i_p_y) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_p_x,*o_p_y).
¬†

`void` **`ObjectTransformUntransformRect`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_r_t, Number i_r_l, Number i_r_b, Number i_r_r, NumberVariable o_r_t, NumberVariable o_r_l, NumberVariable o_r_b, NumberVariable o_r_r )`
Transforms the point (i_r_t,i_r_l,i_r_b,i_r_r) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_r_t,*o_r_l,*o_r_b,*o_r_r).
¬†

`String` **`Octal`**`( Number n )`
Returns the number as an octal string.
¬†

`String` **`Octal`**`( Number n, Number length )`
Returns the number the number as an octal string of the given length.
¬†

`ComplexImageExpression` **`Offset`**`( ComplexImage, Number deltax, Number deltay )`
Returns the pixel with the offset specified by [dh,dv] from the current evaluation position within the image.
¬†

`RGBImageExpression` **`Offset`**`( RGBImage, Number deltax, Number deltay )`
Returns the pixel with the offset specified by [dh,dv] from the current evaluation position within the image.
¬†

`RealImageExpression` **`Offset`**`( RealImage, Number dh, Number dv )`
Returns the pixel with the offset specified by [dh,dv] from the current evaluation position within the image.
¬†

`void` **`OffsetAnnotation`**`( ImageReference, Number annotationID, Number deltax, Number deltay )`
Offset the annotation indicated by the annotationID within the image by [deltax, deltay].

¬†

`Boolean` **`OkCancelDialog`**`( String` *`prompt`* `)`

Puts up a dialog with the given prompt. Returns 1 for OK and 0 for Cancel.

¬†

`void` **`OkDialog`**`( String` *`prompt`* `)`

Puts up a dialog with the given prompt.

¬†

`void` **`OpenAndSetProgressWindow`**`( String` *`line1`*`, String` *`line2`*`, String` *`line3`* `)`

Open the progress window and sets the text within to line1, line2, and line3.

¬†

`Boolean` **`OpenDialog`**`( String` *`pathname`* `)`

Puts up an Open dialog, allows the user to select a file, and stores the pathname into the pathname variable. Returns 1 for OK and 0 for Cancel.

¬†

`Number` **`OpenFileForReading`**`( String` *`fileName`* `)`

Open the file for reading. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.

¬†

`Number` **`OpenFileForReadingAndWriting`**`( String` *`fileName`* `)`

Open the file for reading and writing. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.

¬†

`Number` **`OpenFileForWriting`**`( String` *`fileName`* `)`

Open the file for writing. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.

¬†

`ImageReference` **`OpenImage`**`( String` *`fileName`* `)`

Open the image with the filename. Returns the opened image.

¬†

`void` **`OpenResultsWindow`**`( )`

Open the results window if it is not already open.

¬†

`void` **`OpenTimeBar`**`( String` *`prompt`*`, Number` *`total`* `)`

Opens the time bar with the given prompt. The total parameter is ignored. CloseTimeBar() must be invoked exactly once for every OpenTimeBar() call.

¬†

`Boolean` **`OptionDown`**`( )`

Returns 1 if the option key is down and 0 otherwise.

¬†

`ComplexImage` **`PackedFFT`**`( RealImage` *`source`* `)`

Creates a new packed complex image from the FFT of the real image source.

¬†

`RealImage` **`PackedIFFT`**`( ComplexImage` *`source`* `)`

Creates a new real image from the inverse FFT of the packed complex image source.

¬†

`ComplexImage` **`PackedToComplex`**`( ComplexImage` *`source`* `)`

Creates a new complex 8-byte image from the packed complex source.

¬†

`String` **`PathAddParentIndirection`**`( String` *`path`* `)`

Returns 'path' appended with a string denoting indirection to the parent directory

¬†

`String` **`PathBeginRelative`**`( )`

Returns a string that begins a relative path

¬†

`String` **`PathConcatenate`**`( String` *`initial_path`*`, String` *`final_path`* `)`

Concatenates 'final_path' to 'initial_path' to create a new path, adding separators as necessary.

¬†

`String` **`PathExtractBaseName`**`( String` *`path`*`, Number` *`path_type`* `)`

Returns the base name portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths

¬†

`String` **`PathExtractDirectory`**`( String` *`path`*`, Number` *`path_type`* `)`

Returns the directory portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths

¬†

`String` **`PathExtractExtension`**`( String` *`path`*`, Number` *`path_type`* `)`

Returns the extension portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

`String` **`PathExtractFileName`**`( String` *`path`*`, Number` *`path_type`* `)`
Returns the file name portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

`String` **`PathExtractParentDirectory`**`( String` *`path`*`, Number` *`path_type`* `)`
Returns the parent directory portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

`String` **`PathGetFullpath`**`( String` *`path`* `)`
Returns the full path name of the file denoted by 'dir_path'
¬†

`RealNumberExpression` **`Phase`**`( ComplexNumberExpression )`
Return the phase of the complex number.
¬†

`Number` **`Pi`**`( )`
Return an approximation of pi.
¬†

`void` **`PictureAnnotationSetPicture`**`( Component` *`comp`*`, Number` *`picture`* `)`
Sets the picture of an annotation.
¬†

`void` **`PIctureGetBounds`**`( Number` *`picture`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the preferred bounds of the picture for display on the screen.
¬†

`RealNumberExpression` **`PoissonRandom`**`( Number )`
Return a random number with poisson distribution between [0,1).
¬†

`ComplexNumberExpression` **`Polar`**`( ComplexNumberExpression )`
Return the polar representation of the rectangular complex number.
¬†

`ComplexNumber` **`Polynomial`**`( ComplexImageExpression, ComplexNumber` *`x`* `)`
Return the polynomial a0 + a1*x + a2*x*x ... where x is the free variable and a0 ... aN are the pixels of the image.
¬†

`Number` **`Polynomial`**`( RealImageExpression, Number` *`x`* `)`
Return the polynomial a0 + a1*x + a2*x*x ... where x is the free variable and a0 ... aN are the pixels of the image.
¬†

`void` **`PrintImage`**`( ImageReference )`
Print the image.
¬†

`Number` **`product`**`( RealImageExpression )`
Return the product of all pixel values of the image expression.
¬†

`ComplexNumber` **`product`**`( ComplexImageExpression )`
Return the product of all pixel values of the image expression.
¬†

`RealNumberExpression` **`random`**`( )`
Return a random number in the range 0 to 65535.
¬†

`RasterImageDisplay` **`RasterImageDisplayNullify`**`( ! )`
¬†

`RealImage` **`RasterizeRGB`**`( RGBImage` *`source`*`, Boolean` *`dither`* `)`
Rasterize the source RGB image and return the resulting Raster image displayed image with an appropriate color table. The dither parameter controls dithering.
¬†

`void` **`RawCopyImage`**`( ImageReference` *`src`*`, ImageReference` *`dst`* `)`
Copies the src image to the dst image ignoring data types. The data type sizes of the two images must be the same.
¬†

`String` **`ReadFile`**`( Number` *`file`*`, Number` *`encoding`*`, Number` *`count`* `)`
Reads count bytes from a file, returning them as a string assuming they have the specified encoding.
¬†

`String` **`ReadFile`**`( Number` *`file`*`, Number` *`count`* `)`
Read count bytes from the file, returning them as a string.
¬†

`Boolean` **`ReadFileLine`**`( Number` *`file`*`, String` *`string`* `)`

Read a line of text from the file, storing it into the string variable. Return 1 if successful and 0 otherwise.
¬†

`Boolean` **`ReadFileLine`**`( Number` *`file`*`, Number` *`encoding`*`, String` *`string`*` )`
Read a line of text from the file, storing it into the string variable, assuming the text has the specified encoding. Return 1 if successful and 0 otherwise.
¬†

`void` **`ReadRawStream`**`( Number` *`rawStream`*`, Number` *`data`*`, Number` *`length`*` )`
Read length bytes from rawStream and store it into the memory pointed to by data.
¬†

`RealNumberExpression` **`real`**`( ComplexNumberExpression )`
Return the real portion of the complex number.
¬†

`ComplexImage` **`RealFFT`**`( RealImage` *`source`*` )`
Creates a new complex 8-byte image from the FFT of the real image source.
¬†

`RealImage` **`RealIFFT`**`( ComplexImage` *`source`*` )`
Creates a new real image from the inverse FFT of the complex 8-byte image source.
¬†

`RealImage` **`RealImage`**`( String` *`title`*`, Number` *`bytes`*`, Number` *`d0`*` )`
Creates a 1D real image of size [d0] with the given title. The bytes parameter can be 4 or 8 for single and double precision floating point numbers.
¬†

`RealImage` **`RealImage`**`( String` *`title`*`, Number` *`bytes`*`, Number` *`d0`*`, Number` *`d1`*` )`
Creates a 2D real image of size [d0,d1] with the given title. The bytes parameter can be 4 or 8 for single and double precision floating point numbers.
¬†

`RealImage` **`RealImage`**`( String` *`title`*`, Number` *`bytes`*`, Number` *`d0`*`, Number` *`d1`*`, Number` *`d2`*` )`
Creates a 3D real image of size [d0,d1,d2] with the given title. The bytes parameter can be 4 or 8 for single and double precision floating point numbers.
¬†

`ComplexNumberExpression` **`Rect`**`( ComplexNumberExpression )`
Return the rectangular representation of the polar complex number.
¬†

`RealNumberExpression` **`red`**`( RGBNumberExpression )`
Return the red portion of an RGB number.
¬†

`void` **`Reduce`**`( ImageReference )`
Reduces the image by 2X.
¬†

`ComplexImage` **`ReducedFFT`**`( ImageReference` *`source`*` )`
Creates a new packed complex image from the FFT of the real image source after reducing the source by a factor of 2.
¬†

`void` **`RegisterClass`**`( String` *`class_name`*`, String` *`parent_class_name`*`, ! alloc_proc, ! dealloc_proc, Number` *`refCon`*` )`
¬†

`Number` **`RegisterCustomMenu`**`( Number` *`menuHandler`*` )`
Register a custom menu. See the SDK documentation for more information.
¬†

`void` **`RegisterMenuAdjustment`**`( String` *`menuName`*`, String` *`adjustedMenuName`*` )`
Unregister a menu name adjustment. See the SDK documentation for more information.
¬†

`Number` **`RegisterObjectListener`**`( Number` *`object`*`, Number` *`proc`*`, Number` *`refCon`*` )`
Add object listener to OM object. See the SDK documentation for more information.
¬†

`void` **`RegisterScriptPalette`**`( ScriptObject, String` *`type`*`, String` *`name`*` )`
¬†

`void` **`ReleaseSemaphore`**`( Number )`
Release a semaphore. Used only with background processing.
¬†

`RealNumberExpression` **`remainder`**`( RealNumberExpression, RealNumberExpression )`
Return the remainder of the first number divided by the second number.
¬†

`void` **`RemovePathFromCopyToImageList`**`( String` *`path`*` )`
Removes 'path' from the copy to image list.

¬†

`void` **`RemoveScriptFromMenu`**`( String commandName, String menuName, String optionalSubMenuName )`

Removes the given menu command from the menu. The commandName indicates the string by which this script is known to the application. The menuName and optionalSubMenuName parameters specify the menu.

¬†

`void` **`Result`**`( String )`

Output the string to the results window.

¬†

`void` **`Result`**`( Number, String format )`

Output the real number with the printf-style format to the results window.

¬†

`void` **`Result`**`( Number )`

Output the real number to the results window.

¬†

`void` **`Result`**`( RGBNumber )`

Output the RGB number to the results window.

¬†

`void` **`Result`**`( ComplexNumber )`

Output the complex number to the results window.

¬†

`RGBNumberExpression` **`rgb`**`( RealNumberExpression red, RealNumberExpression green, RealNumberExpression blue )`

Creates an RGB number from the individual number components.

¬†

`RGBNumberExpression` **`rgba`**`( RealNumberExpression red, RealNumberExpression green, RealNumberExpression blue, RealNumberExpression alpha )`

Creates an RGB number from the individual number components, including the alpha channel.

¬†

`RGBImage` **`RGBImage`**`( String title, Number bytes, Number d0, Number d1, Number d2 )`

Creates a 3D RGB image of size [d0,d1,d2] with the given title. The bytes parameter must be 4.

¬†

`RGBImage` **`RGBImage`**`( String title, Number bytes, Number d0, Number d1 )`

Creates a 2D RGB image of size [d0,d1] with the given title. The bytes parameter must be 4.

¬†

`RGBImage` **`RGBImage`**`( String title, Number bytes, Number d0 )`

Creates a 1D RGB image of size [d0] with the given title. The bytes parameter must be 4.

¬†

`String` **`right`**`( String, Number count )`

Returns rightmost count characters of a string.

¬†

`Number` **`RMS`**`( RealImageExpression )`

Return the RMS of all pixel values of the image expression.

¬†

`ROI` **`ROINullify`**`( ! )`

¬†

`RealImage` **`Rotate`**`( ImageReference source, Number radians )`

Creates a new real image by rotating the source image counterclockwise by radians.

¬†

`void` **`RotateLeft`**`( ImageReference )`

Rotates the image to the left by 90¬°.

¬†

`void` **`RotateRight`**`( ImageReference )`

Rotates the image to the right by 90¬°.

¬†

`RealNumberExpression` **`Round`**`( RealNumberExpression )`

Return the number converted to integer using current rounding mode.

¬†

`void` **`Save`**`( ImageReference )`

Save the image under the it's current filename.

¬†

`Boolean` **`SaveAsDialog`**`( String prompt, String defaultName, String saveName )`

Puts up the SaveAs dialog with the given prompt and default file name, and then stores the chosen path for the saved file into the saveName variable. Returns 1 for OK and 0 for Cancel.

¬†

`void` **`SaveAsGatan`**`( ImageReference, String fileName )`

Save the image to the fileName in Gatan 3.0 file format.
¬†
void **SaveAsGIF**(ImageReference, String *fileName*)
Save the image to the fileName in GIF file format. The DM Import/Export Plug-in must be installed in order for this function to work.
¬†
void **SaveAsPCX**(ImageReference, String *fileName*)
Save the image to the fileName in PCX file format. The DM Import/Export Plug-in must be installed in order for this function to work.
¬†
void **SaveAsPICT**(ImageReference, String *fileName*)
Save the image to the fileName in PICT file format. The DM Import/Export Plug-in must be installed in order for this function to work.
¬†
void **SaveAsRawData**(ImageReference, String *fileName*)
Save the image to the fileName in raw file format.
¬†
void **SaveAsSmallHeader**(ImageReference, String *fileName*)
Save the image to the fileName in Gatan small header file format.
¬†
void **SaveAsText**(ImageReference, String *fileName*)
Save the image to the fileName in text file format.
¬†
void **SaveAsTIFF**(ImageReference, String *fileName*)
Save the image to the fileName in TIFF file format. The DM Import/Export Plug-in must be installed in order for this function to work.
¬†
void **SaveImage**(ImageReference, String *fileName*)
Save the image to the fileName in it's current file format.
¬†
void **SaveNotes**(ImageReference, String *tagPath*, String *fileName*)
Saves the image notes indicated by tagPath into the given fileName. The tagPath must refer to a group.
¬†
void **SavePersistentNotes**(String *tagPath*, String *fileName*)
Saves the persistent notes indicated by tagPath into the given fileName. The tagPath must refer to a group.
¬†
void **ScrapClear**(ImageReference)
Clear the pasted image (if there is one) from the image.
¬†
void **ScrapCopy**(ImageReference)
Copy the selected portion of the image to the scrap.
¬†
void **ScrapGetLocation**(ImageReference, NumberVariable *top*, NumberVariable *left*)
Store the pasted image location (in image coordinates) into the top and left variables.
¬†
void **ScrapGetSize**(ImageReference, NumberVariable *width*, NumberVariable *height*)
Store the pasted image size (in image coordinates) into the width and height variables.
¬†
void **ScrapMerge**(ImageReference)
Merge the pasted image (if there is one) with the image.
¬†
void **ScrapPaste**(ImageReference)
Paste the scrap into the image.
¬†
void **ScrapPasteNew**( )
Paste the scrap into a new image.
¬†
void **ScrapSetLocation**(ImageReference, Number *top*, Number *left*)
Set the location of the pasted image (if there is one) to [top,left].
¬†
void **ScreenGetBounds**(Number *index*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the bounds of the 'index'th screen.
¬†
void **ScreenGetWorkArea**(Number *index*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the bounds of the 'index'th screen.
¬†
void **ScriptInterfaceGenerateStubs**(String *interface_name*, String *str_h*, String *str_cp*)

Generates C++ stubs for the interface.
¬†
`ScriptObject` **`ScriptObjectNullify`**`( ! )`
¬†
`String` **`ScriptToUserCoordinates`**`( ImageReference, Number` *`x`*`, Number` *`y`* `)`
Return a string representing the script coordinates.
¬†
`void` **`ScriptWindowExecute`**`( DocumentWindow` *`window`* `)`
Executes the script in the script window.
¬†
`void` **`SelectAnnotation`**`( ImageReference, Number` *`annotationID`* `)`
Select the annotation indicated by the annotationID within the image.
¬†
`void` **`SelectImage`**`( ImageReference )`
Bring the image's image document window to the front.
¬†
`void` **`SetAnnotationBackground`**`( ImageReference, Number` *`annotationID`*`, Number` *`background`* `)`
Set the background mode of the annotation indicated by the annotationID within the image. Possible values for background are 1=black-on-white, 2=black, 3=white-on-black, 4=white.
¬†
`void` **`SetAnnotationComplexNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, ComplexNumber` *`noteValue`* `)`
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
`void` **`SetAnnotationFace`**`( ImageReference, Number` *`annotationID`*`, Number` *`face`* `)`
Set the type face of the annotation indicated by the annotationID within the image. Values for the face can be found in InsideMacintosh Vol I.
¬†
`void` **`SetAnnotationFont`**`( ImageReference, Number` *`annotationID`*`, String` *`fontName`* `)`
Set the font of the annotation indicated by the annotationID within the image to fontname.
¬†
`void` **`SetAnnotationJustification`**`( ImageReference, Number` *`annotationID`*`, Number` *`justification`* `)`
Set the justification of the text annotation indicated by the annotationID within the image. Possible values for justification are -1=left, 0=center, 1=right.
¬†
`void` **`SetAnnotationNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, Number` *`noteValue`* `)`
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
`void` **`SetAnnotationRect`**`( ImageReference, Number` *`annotationID`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Set the bounds of the annotation indicated by the annotationID within the image to [top, left, bottom, right].
¬†
`void` **`SetAnnotationRectNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Sets the value of the annotation note with label noteLabel to the rectangle formed from the top, left, bottom, and right parameters. If the note does not exist, it is created.
¬†
`void` **`SetAnnotationRGBNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, RGBNumber` *`noteValue`* `)`
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
`void` **`SetAnnotationSize`**`( ImageReference, Number` *`annotationID`*`, Number` *`size`* `)`
Set the size of text of the annotation indicated by the annotationID within the image.
¬†
`void` **`SetAnnotationStringNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, String` *`noteValue`* `)`
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
`void` **`SetApplicationFont`**`( Number` *`which_font`*`, String` *`face_name`*`, Number` *`attributes`*`, Number` *`size`*`, Number` *`text_encoding`* `)`
Sets the selected appliction font to have the specified face name, attributes, size, and text encoding.
¬†
`void` **`SetColorMode`**`( ImageReference, Number` *`mode`* `)`
Set the color mode of the image. The possible values for mode are 1=greyscale, 3=rainbow, 4=temperature.
¬†

`void` **SetComplexMode**(ImageReference, Number *mode*)
Set the complex-to-real mode of the image. The possible value for mode are 1=real, 2=imaginary, 3=modulus, 4=log of modulus, 5=phase.
¬†

`void` **SetComplexNumberNote**(ImageReference, String *noteLabel*, ComplexNumber *noteValue*)
Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†

`void` **SetContrastMode**(ImageReference, Number *mode*)
Set the constrast mode of the image. The possible value for mode are 1=linear, 2=equalized, 3=contoured, 4=custom.
¬†

`void` **SetCustomCLUT**(ImageReference, RGBImage *clutImage*)
Set the CLUT for the image to clutImage. clutImage must be a 256x1 RGB image.
¬†

`void` **SetDisplayType**(ImageReference, Number *displayType*)
Set the display type of the image. Possible displayType values are -1=best, 1=raster image, 2=surface plot, 3=RGB, 4=line plot, 5=spreadsheet.
¬†

`void` **SetDoCreateCustomIcon**(ImageReference *image*, Boolean *doCreateCustomIcon*)
Open the image with the filename. Returns the opened image.
¬†

`void` **SetDoSavePreview**(ImageReference *image*, Boolean *doSavePreview*)
Sets whether to save a preview along with the saved file.
¬†

`void` **SetEstimatedMinMax**(ImageReference, Number *min*, Number *max*)
Set the estimated minimum and maximum of the image to min, max. This can be used if survey is turned off.
¬†

`void` **SetImage**(ImageReference *src*, RealImageVariable *dst*)
Assigns src to dst by reference.
¬†

`void` **SetImagePositionWithinWindow**(ImageReference, Number *x*, Number *y*)
Set top-left position of the image to [x,y] within the image document.
¬†

`void` **SetInversionMode**(ImageReference, Boolean *inverted*)
Set the contrast of the image to be inverted or not inverted.
¬†

`void` **SetKeywordNote**(ImageReference, String *keyword*)
Adds the keyword to the image keyword list if it does not already exist.
¬†

`void` **SetLimits**(ImageReference, Number *low*, Number *high*)
Set the lowest and highest displayed pixel values for the image. Everything below low will be the 'black' color and every above high will be the 'white' color. The black and white colors may not actually be black and white if the color table is not greyscale.
¬†

`void` **SetMinContrast**(ImageReference, Number *minContrast*)
Set the minimum amount of contrast for the image to minContrast.
¬†

`void` **SetName**(ImageReference, String *name*)
Sets the name of the image's image document to name.
¬†

`void` **SetNoteState**(ImageReference, String *noteLabel*, Boolean *hidden*)
Sets whether the given image note is in the copy-to-image list. The hidden parameter is unused.
¬†

`void` **SetNumberNote**(ImageReference, String *noteLabel*, Number *noteValue*)
Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†

`void` **SetOrigin**(ImageReference, Number *x*, Number *y*)
Set the origin of the image coordinates to [x,y] which are in pixel units.
¬†

`void` **SetPersistentComplexNumberNote**(String *noteLabel*, ComplexNumber *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†

`void` **SetPersistentKeywordNote**(String *keyword*)
Adds the keyword to the persistent keyword list if it does not already exist.
¬†

`void` **SetPersistentLongNote**(String *noteLabel*, Number *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.

¬†
void **SetPersistentNoteState**(String *noteLabel*, Boolean *copyToImage*, Boolean *hidden*)
Sets whether the given persistent note is in the copy-to-image list. The hidden parameter is unused.
¬†
void **SetPersistentNumberNote**(String *noteLabel*, Number *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPersistentPointNote**(String *noteLabel*, Number *x*, Number *y*)
Sets the value of the persistent note with label noteLabel to the point formed from the x and y parameters. If the note does not exist, it is created.
¬†
void **SetPersistentRectNote**(String *noteLabel*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the value of the persistent note with label noteLabel to the rectangle formed from the top, left, bottom, and right parameters. If the note does not exist, it is created.
¬†
void **SetPersistentRGBNumberNote**(String *noteLabel*, RGBNumber *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPersistentStringNote**(String *noteLabel*, String *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPixel**(RGBImage, Number *x*, Number *y*, RGBNumber *value*)
Sets the pixel in the image at [x,y] to value.
¬†
void **SetPixel**(ComplexImage, Number *x*, Number *y*, ComplexNumber *value*)
Sets the pixel in the image at [x,y] to value. Does not work on packed complex images.
¬†
void **SetPixel**(RealImage, Number *x*, Number *y*, Number *value*)
Sets the pixel in the image at [x,y] to value.
¬†
void **SetPointNote**(ImageReference, String *noteLabel*, Number *x*, Number *y*)
Sets the value of the image note with label noteLabel to the point formed from the x and y parameters. If the note does not exist, it is created.
¬†
void **SetRawStreamPos**(Number *rawStream*, Number *mode*, Number *offset*)
Set the current position in rawStream to offset using the mode.
¬†
void **SetRectNote**(ImageReference, String *noteLabel*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the value of the image note with label noteLabel to the rectangle formed from the top, left, bottom, and right parameters. If the note does not exist, it is created.
¬†
void **SetRGBNumberNote**(ImageReference, String *noteLabel*, RGBNumber *noteValue*)
Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetScale**(ImageReference, Number *x*, Number *y*)
Set the scale of the image to [x,y].
¬†
void **SetSelection**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the selection of the image to [top,left,bottom,right].
¬†
void **SetStringNote**(ImageReference, String *noteLabel*, String *noteValue*)
Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetSurvey**(ImageReference, Boolean *surveyOnOff*)
Turn surveying on or off for the image.
¬†
void **SetSurveyTechnique**(ImageReference, Number *mode*)
Set the survey technique for the image to mode. Mode can be 0=cross-wire, 1=whole image, 2=sparse.
¬†
void **SetUnitString**(ImageReference, String *unitString*)
Set the units of the image to unitString.
¬†
void **SetWindowBounds**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)

Set the image's image document window bounds to (left,top),(right,bottom).
¬†
`void` **`SetWindowPosition`**`( ImageReference, Number` *`xPos`*`, Number` *`yPos`* `)`
Set the image's image document window position to [xPos, yPos]. Only valid for images that are already shown in a window.
¬†
`void` **`SetWindowSize`**`( ImageReference, Number` *`width`*`, Number` *`height`* `)`
Set the image's image document window size to [width, height].
¬†
`void` **`SetZoom`**`( ImageReference, Number` *`zoom`* `)`
Set the zoom of the image display.
¬†
`RealNumberExpression` **`sgn`**`( RealNumberExpression )`
Returns 1 if the number is equal to or greater than 0 otherwise returns -1.
¬†
`void` **`ShiftCenter`**`( ImageReference` *`image`* `)`
Shifts each dimension of an image by half. For two dimensional images it will swap quadrants.
¬†
`Boolean` **`ShiftDown`**`( )`
Returns 1 if the shift key is down and 0 otherwise.
¬†
`void` **`ShowAlert`**`( String` *`prompt`*`, Number` *`alertStyle`* `)`
Puts up an alert with the given prompt and style.
¬†
`void` **`ShowImage`**`( ImageReference )`
Display the image's image document in a window if it is not displayed already.
¬†
`ComplexNumberExpression` **`sin`**`( ComplexNumberExpression )`
Return the sine of the number.
¬†
`RealNumberExpression` **`sin`**`( RealNumberExpression )`
Return the sine of the number.
¬†
`ComplexNumberExpression` **`sinh`**`( ComplexNumberExpression )`
Return the hyperbolic sine of the number.
¬†
`RealNumberExpression` **`sinh`**`( RealNumberExpression )`
Return the hyberbolic sine of the number.
¬†
`void` **`Sleep`**`( Number` *`seconds`* `)`
Puts the current thread to sleep for the given number of seconds (resolution may vary by platform).
¬†
`BasicImage` **`slice1`**`( BasicImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
`RGBImage` **`slice1`**`( RGBImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
`RealImage` **`slice1`**`( RealImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
`ComplexImage` **`slice1`**`( ComplexImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
`BasicImage` **`slice2`**`( BasicImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`*`, Number` *`d1`*`, Number` *`l1`*`, Number` *`s1`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
`RealImage` **`slice2`**`( RealImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`*`, Number` *`d1`*`, Number` *`l1`*`, Number` *`s1`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
`ComplexImage` **`slice2`**`( ComplexImage, Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, Number` *`d0`*`, Number` *`l0`*`, Number` *`s0`*`, Number` *`d1`*`, Number` *`l1`*`, Number` *`s1`* `)`
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†

RGBImage **slice2**(RGBImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
RGBImage **slice3**(RGBImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
BasicImage **slice3**(BasicImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
ComplexImage **slice3**(ComplexImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
RealImage **slice3**(RealImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
Boolean **SpaceDown**( )
Returns 1 if the space key is down and 0 otherwise.
¬†
RealNumberExpression **SphericalBesselJ**(Number *order*, Number)
Return the spherical Bessel J function of the given order for a number.
¬†
RealNumberExpression **SphericalBesselY**(Number, Number)
Return the spherical Bessel Y function of the given order for a number.
¬†
ComplexNumberExpression **sqrt**(ComplexNumberExpression)
Return the square root of the number.
¬†
RealNumberExpression **sqrt**(RealNumberExpression)
Return the square root of the number.
¬†
void **StopAcquisitionDaemon**(ImageReference)
Stop all acquisition daemons associated with the image.
¬†
void **StopAllAcquisitionDaemons**( )
Stop all acquisition daemons with any image within the application.
¬†
Number **StreamGetPos**(ScriptObject *stream*)
Returns the current position within the file.
¬†
Number **StreamGetSize**(ScriptObject *stream*)
Returns the size of the file.
¬†
String **StreamReadAsText**(ScriptObject *stream*, Number *encoding*, Number *count*)
Reads count bytes from a file, returning them as a string assuming they have the specified encoding.
¬†
Boolean **StreamReadTextLine**(ScriptObject *stream*, Number *encoding*, String *str_out*)
Read a line of text from the stream, storing it into the string variable, assuming the text has the specified encoding. Return 1 if successful and 0 otherwise.
¬†
void **StreamSetPos**(ScriptObject *stream*, Number *base*, Number *offset*)
Sets the current position within the file as an offset from 'base', where base == 0 denotes beginning of file, 1 denotes current position, and 2 denotes end of file.
¬†
void **StreamSetSize**(ScriptObject *stream*, Number *size*)
Sets the size of the file.
¬†
void **StreamWriteAsText**(ScriptObject *stream*, Number *encoding*, String *data*)
Write the string to the file with the specified encoding.
¬†
String **StringNullify**(!)

Assigns NULL to dst string.
¬†
`ComplexNumber` **`sum`**`( ComplexImageExpression )`
Return the sum of all pixel values of the image expression.
¬†
`Number` **`sum`**`( RealImageExpression )`
Return the sum of all pixel values of the image expression.
¬†
`SurfacePlotImageDisplay` **`SurfacePlotImageDisplayNullify`**`( ! )`
¬†
`RealImage` **`SVDecomposition`**`( RealImage` *`a`*`, RealImage` *`b`*` )`
Return the image resulting from a SV decomposition on images a,b.
¬†
`RealImage` **`SVDFit`**`( RealImage` *`a`*`, RealImage` *`b`*`, Number` *`tolerance`*` )`
Return the image resulting from a SVD fit on images a,b with the given tolerance.
¬†
`void` **`swap`**`( RealImageExpression` *`a`*`, RealImageExpression` *`b`*` )`
Swap images a and b, pixel by pixel.
¬†
`void` **`swap`**`( ComplexImageExpression` *`a`*`, ComplexImageExpression` *`b`*` )`
Swap images a and b, pixel by pixel.
¬†
`void` **`SwapByteOrder`**`( ImageReference )`
Swaps the byte order for each long word in the image. ABCD become DCBA.
¬†
`void` **`SwapWordOrder`**`( ImageReference )`
Swaps the byte order for each short word in the image. ABCD become BADC.
¬†
`TagGroup` **`TagGroupNullify`**`( ! )`
¬†
`RealNumberExpression` **`tan`**`( RealNumberExpression )`
Return the tangent of the number.
¬†
`ComplexNumberExpression` **`tan`**`( ComplexNumberExpression )`
Return the tangent of the number.
¬†
`RealNumberExpression` **`tanh`**`( RealNumberExpression )`
Return the hyperbolic tangent of the number.
¬†
`ComplexNumberExpression` **`tanh`**`( ComplexNumberExpression )`
Return the hyperbolic tangent of the number.
¬†
`RealNumberExpression` **`tert`**`( RealNumberExpression` *`condition`*`, RealNumberExpression` *`truenumber`*`, RealNumberExpression` *`falsenumber`*` )`
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
`ComplexNumberExpression` **`tert`**`( RealNumberExpression` *`condition`*`, ComplexNumberExpression` *`truenumber`*`, ComplexNumberExpression` *`falsenumber`*` )`
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
`RGBNumberExpression` **`tert`**`( RealNumberExpression` *`condition`*`, RGBNumberExpression` *`truenumber`*`, RGBNumberExpression` *`falsenumber`*` )`
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
`void` **`Test_AssignUniformRandom`**`( ImageReference` *`img`*` )`
¬†
`Number` **`Test_IncCounter`**`( Number` *`period`*` )`
¬†
`TagGroup` **`Test_ReferenceCount_V1`**`( ROI` *`roi_out`*` )`
¬†
`TagGroup` **`Test_ReferenceCount_V2`**`( ROI` *`roi_out`*` )`
¬†
`void` **`Test_ResetCounter`**`( )`
¬†
`Number` **`TextAnnotationGetAlignment`**`( Component` *`comp`*` )`

Gets the alignment of the text in the text annotation.
¬†
`void` **`TextAnnotationGetFixedPoint`**`( Component` *`comp`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Gets the fixed point of the text annotation.
¬†
`Number` **`TextAnnotationGetResizeStyle`**`( Component` *`comp`*` )`
Gets the resize style of the text annotation.
¬†
`String` **`TextAnnotationGetText`**`( Component` *`comp`*` )`
Gets the text of a text annotation.
¬†
`void` **`TextAnnotationSetAlignment`**`( Component` *`comp`*`, Number` *`alignment`*` )`
Sets the alignment of the text in the text annotation.
¬†
`void` **`TextAnnotationSetFixedPoint`**`( Component` *`comp`*`, Number` *`x`*`, Number` *`y`*` )`
Sets the fixed point of the text annotation.
¬†
`void` **`TextAnnotationSetResizeStyle`**`( Component` *`comp`*`, Number` *`style`*` )`
Sets the resize style of the text annotation.
¬†
`void` **`TextAnnotationSetText`**`( Component` *`comp`*`, String` *`text`*` )`
Sets the text of a text annotation.
¬†
`void` **`Throw`**`( Number` *`exception`*` )`
Throws an exception by number.
¬†
`void` **`Throw`**`( String` *`exception`*` )`
Throws an exception by string.
¬†
`Throwable` **`ThrowableNullify`**`( ! )`
Assigns NULL to dst throwable.
¬†
`Number` **`TickCount`**`( )`
Return the MacOS system tick count.
¬†
`RGBImageExpression` **`TimeBar`**`( String` *`title`*`, RGBImageExpression` *`expression`*` )`
Displays a timebar with the given title during evaluation of the expression. Return the expression directly.
¬†
`ComplexImageExpression` **`TimeBar`**`( String` *`title`*`, ComplexImageExpression` *`expression`*` )`
Displays a timebar with the given title during evaluation of the expression. Return the expression directly.
¬†
`RealImageExpression` **`TimeBar`**`( String` *`title`*`, RealImageExpression` *`expression`*` )`
Displays a timebar with the given title during evaluation of the expression. Return the expression directly.
¬†
`void` **`TransferTagsAndApplyDataBar`**`( ImageReference )`
Transfer tags and apply data bar to the image.
¬†
`void` **`TransformPointFromImageToWindow`**`( ImageReference` *`image`*`, Number` *`x_image`*`, Number` *`y_image`*`, NumberVariable` *`x_window`*`, NumberVariable` *`y_window`*` )`
Place in (x_window,y_window) the
¬†
`RealNumberExpression` **`Trunc`**`( RealNumberExpression )`
Return the number truncated to an integer (rounding towards zero).
¬†
`ImageReference` **`TryToUse`**`( String` *`name`*`, Number` *`width`*`, Number` *`height`*`, Number` *`dataType`*`, Number` *`h`*`, Number` *`v`*`, NumberVariable` *`fresh`*` )`
Look for an image with the given name, width, height, dataType, and position [h,v]. Return it if found and create it if not. Store 1 into fresh if it was created and 0 if not.
¬†
`Boolean` **`TwoButtonDialog`**`( String` *`prompt`*`, String` *`acceptLabel`*`, String` *`rejectLabel`*` )`
Puts up a dialog with the given prompt and two buttons labeled according to the parameters. Returns 1 for the acceptLabel button and false for the other one.
¬†
`Number` **`unc`**`( String, Number` *`index`*` )`
Returns the unicode value of the first character of the string.

¬†

`RealNumberExpression` **`UniformRandom`**`( )`

Return a random number with uniform distribution between [0,1).

¬†

`void` **`UnregisterClass`**`( String` *`class_name`* `)`

¬†

`void` **`UnregisterCustomMenu`**`( Number` *`menuHandlerToken`* `)`

Unregister a custom menu. See the SDK documentation for more information.

¬†

`void` **`UnregisterObjectListener`**`( Number` *`object`*`, Number` *`id`* `)`

Remove object listener from OM object. See the SDK documentation for more information.

¬†

`void` **`UpdateDisplay`**`( ImageReference, Number` *`startRow`*`, Number` *`endRow`* `)`

Update the image display portion that displays the image rows from startRow to endRow.

¬†

`void` **`UpdateImage`**`( ImageReference )`

Update the image immediately.

¬†

`void` **`UpdatePictureAnnotation`**`( ImageReference, Number` *`annotationID`*`, ! PicHandle )`

Update the picture annotation indicated by the annotationID within the image with the new picHandle.

¬†

`void` **`UpdateTimeBarPercentage`**`( Number` *`percentage`* `)`

Updates an open time bar to the given percentage.

¬†

`Number` **`val`**`( String )`

Returns the numeric value of a string.

¬†

`Boolean` **`ValidAnnotation`**`( ImageReference, Number` *`annotationID`* `)`

Return 1 if the annotationID is valid; returns 0 otherwise.

¬†

`Number` **`variance`**`( ImageReference )`

Return the variance of the image.

¬†

`ComplexImageExpression` **`Warp`**`( ComplexImage` *`source`*`, RealImageExpression` *`sourceX`*`, RealImageExpression` *`sourceY`* `)`

Returns the bilinear interpolated value at the position [x,y] within the source image.

¬†

`RealImageExpression` **`Warp`**`( RealImage` *`source`*`, RealImageExpression` *`x`*`, RealImageExpression` *`y`* `)`

Returns the bilinear interpolated value at the position [x,y] within the source image.

¬†

`RGBImageExpression` **`Warp`**`( RGBImage` *`source`*`, RealImageExpression` *`sourceX`*`, RealImageExpression` *`sourceY`* `)`

Returns the bilinear interpolated value at the position [x,y] within the source image.

¬†

`DocumentWindow` **`WindowNullify`**`( ! )`

¬†

`void` **`WriteFile`**`( Number` *`file`*`, Number` *`encoding`*`, String` *`data`* `)`

Write the string to the file with the specified encoding.

¬†

`void` **`WriteFile`**`( Number` *`file`*`, String` *`data`* `)`

Write the string to the file.

¬†

`void` **`WriteRawStream`**`( Number` *`rawStream`*`, Number` *`data`*`, Number` *`length`* `)`

Write length bytes from the memory pointed to by data to rawStream.

¬†

`Number` **`XX_GetProcessKey`**`( Number` *`pid`* `)`

Return a keystroke associated with the background process indicated by pid.

¬†

`void` **`XX_InstallImageProcess`**`( ImageReference, Number` *`pid`* `)`

Associate the process indicated by pid with the image. Keystrokes going to the image will queue in the process after this call.

¬†

`void` **`Yield`**`( )`

Yield to another background task.

¬†

DigitalMicrograph Script Functions

Return to Top

`RealNumberExpression !( RealNumberExpression src )`
Return the logical NOT of src.
¬†
`Boolean !( Boolean v1 )`
¬†
`Boolean !=( String, String )`
Returns true if the two strings are not equal character-by-character.
¬†
`RealNumberExpression !=( RGBNumberExpression, RGBNumberExpression )`
Return 1 if the two RGB values are not equal and 0 otherwise.
¬†
`RealNumberExpression !=( ComplexNumberExpression, ComplexNumberExpression )`
Return 1 if the two numbers are not equal and 0 otherwise.
¬†
`Boolean !=( Number v1, Number v2 )`
¬†
`RealNumberExpression !=( RealNumberExpression, RealNumberExpression )`
Return 1 if the two numbers are not equal and 0 otherwise.
¬†
`Boolean &&( Boolean v1, Boolean v2 )`
¬†
`RealNumberExpression &&( RealNumberExpression, RealNumberExpression )`
Return the logical AND of the two numbers.
¬†
`RealNumberExpression *( RealNumberExpression, RealNumberExpression )`
Return the product of the two numbers.
¬†
`Number *( Number v1, Number v2 )`
¬†
`ComplexNumberExpression *( ComplexNumberExpression, ComplexNumberExpression )`
Return the product of the two numbers.
¬†
`RGBNumberExpression *( RGBNumberExpression, RGBNumberExpression )`
Return the product of the two RGB values.
¬†
`ComplexNumberExpression **( ComplexNumberExpression n, ComplexNumberExpression exp )`
Return n raised to the exp power.
¬†
`Number **( Number v1, Number v2 )`
¬†
`RealNumberExpression **( RealNumberExpression n, RealNumberExpression exp )`
Return n raised to the exp power.
¬†
`RealNumberExpression *=( lvalue RealNumberExpression dst, RealNumberExpression src )`
Assign dst*src to dst.
¬†
`ComplexNumberExpression *=( lvalue ComplexNumberExpression dst, ComplexNumberExpression src )`
Assign dst*src to dst.
¬†
`Number *=( NumberVariable dst, Number src )`
¬†
`String +( ComplexNumber, String )`
Concatenates the string to the complex number and returns the resulting string.
¬†

`String +( String, RGBNumber )`
Concatenates the RGB value to the string and returns the resulting string.
¬†
`String +( RGBNumber, String )`
Concatenates the string to the RGB value and returns the resulting string.
¬†
`String +( String, String )`
Concatenates the string to the other string and returns the resulting string.
¬†
`String +( String, ComplexNumber )`
Concatenates the complex number to the string and returns the resulting string.
¬†
`String +( Number, String )`
Concatenates the string to the number and returns the resulting string.
¬†
`RealNumberExpression +( RealNumberExpression, RealNumberExpression )`
Return the sum of the two numbers.
¬†
`Number +( Number v1, Number v2 )`
¬†
`ComplexNumberExpression +( ComplexNumberExpression, ComplexNumberExpression )`
Return the sum of the two numbers.
¬†
`RGBNumberExpression +( RGBNumberExpression, RGBNumberExpression )`
Return the sum of the two RGB values.
¬†
`String +( String, Number )`
Concatenates the number to the string and returns the resulting string.
¬†
`Number ++( NumberVariable v1 )`
¬†
`RealNumberExpression ++( lvalue RealNumberExpression dst )`
Increment dst by 1.
¬†
`Number ++!( NumberVariable v1 )`
¬†
`RealNumberExpression ++!( lvalue RealNumberExpression dst )`
Increment dst by 1.
¬†
`RealNumberExpression +=( lvalue RealNumberExpression dst, RealNumberExpression src )`
Assign dst+src to dst.
¬†
`ComplexNumberExpression +=( lvalue ComplexNumberExpression dst, ComplexNumberExpression src )`
Assign dst+src to dst.
¬†
`String +=( String s1, String s2 )`
Appends string s2 to s1, converting its encoding to that of s1 if necessary.
¬†
`Number +=( NumberVariable dst, Number src )`
¬†
`RealNumberExpression -( RealNumberExpression, RealNumberExpression )`
Return the difference of the two numbers.
¬†
`RealNumberExpression -( RealNumberExpression )`
Return the negation of the number.
¬†
`Number -( Number v1 )`
¬†
`RGBNumberExpression -( RGBNumberExpression, RGBNumberExpression )`
Return the difference of the two RGB values.
¬†
`RGBNumberExpression -( RGBNumberExpression )`
Return the negation of the RGB value. This is the same as RGBA(255,255,255,255)-value.
¬†
`ComplexNumberExpression -( ComplexNumberExpression, ComplexNumberExpression )`

Return the difference of the two numbers.
¬†
`ComplexNumberExpression -( ComplexNumberExpression )`
Return the negation of the number.
¬†
`Number -( Number v1, Number v2 )`
¬†
`RealNumberExpression --( lvalue RealNumberExpression dst )`
Decrement dst by 1.
¬†
`Number --( NumberVariable v1 )`
¬†
`RealNumberExpression --!( lvalue RealNumberExpression dst )`
Decrement dst by 1.
¬†
`Number --!( NumberVariable v1 )`
¬†
`RealNumberExpression -=( lvalue RealNumberExpression dst, RealNumberExpression src )`
Assign dst-src to dst.
¬†
`Number -=( NumberVariable dst, Number src )`
¬†
`ComplexNumberExpression -=( lvalue ComplexNumberExpression dst, ComplexNumberExpression src )`
Assign dst-src to dst.
¬†
`RGBNumberExpression /( RGBNumberExpression, RGBNumberExpression )`
Return the result of dividing the two RGB values.
¬†
`ComplexNumberExpression /( ComplexNumberExpression, ComplexNumberExpression )`
Return the result of dividing the two numbers.
¬†
`Number /( Number v1, Number v2 )`
¬†
`RealNumberExpression /( RealNumberExpression, RealNumberExpression )`
Return the result of dividing the two numbers.
¬†
`ComplexNumberExpression /=( lvalue ComplexNumberExpression dst, ComplexNumberExpression src )`
Assign dst/src to dst.
¬†
`Number /=( NumberVariable dst, Number src )`
¬†
`RealNumberExpression /=( lvalue RealNumberExpression dst, RealNumberExpression src )`
Assign dst/src to dst.
¬†
`ImageReference :=( ImageVariable image, ! )`
¬†
`void :=( RGBImageVariable, RGBImage )`
¬†
`void :=( ComplexImageVariable, ComplexImage )`
¬†
`void :=( RealImageVariable, RealImage )`
¬†
`void :=( BasicImage dst, BasicImage src )`
Assign src to dst by reference.
¬†
`Boolean <( Number v1, Number v2 )`
¬†
`RealNumberExpression <( RealNumberExpression, RealNumberExpression )`
Return 1 if the left number is less than the right number and 0 otherwise.
¬†
`Boolean <=( Number v1, Number v2 )`
¬†
`RealNumberExpression <=( RealNumberExpression, RealNumberExpression )`
Return 1 if the left number is less or equal to than the right number and 0 otherwise.
¬†

SurfacePlotImageDisplay =(SurfacePlotImageDisplay *dst*, ImageDisplay *id*)

¬†

SurfacePlotImageDisplay =(SurfacePlotImageDisplay *dst*, !)

¬†

SurfacePlotImageDisplay =(SurfacePlotImageDisplay *dst*, SurfacePlotImageDisplay *spid*)

¬†

DocumentWindow =(DocumentWindow, !)

¬†

DocumentWindow =(DocumentWindow, DocumentWindow)

¬†

ImageDocument =(ImageDocument, !)

¬†

ImageDocument =(ImageDocument, ImageDocument)

¬†

ImageDisplay =(ImageDisplay, Component)

¬†

ImageDisplay =(ImageDisplay, !)

¬†

ImageDisplay =(ImageDisplay, ImageDisplay)

¬†

Component =(Component, !)

¬†

Component =(Component, Component)

¬†

RGBImageExpression =(lvalue RGBImageVariable, RGBImageExpression)

¬†

ComplexImageExpression =(lvalue ComplexImageVariable, ComplexImageExpression)

¬†

RealImageExpression =(lvalue RealImageVariable, RealImageExpression)

¬†

RGBImageExpression =(lvalue RGBImageExpression, RGBImageExpression)

¬†

ComplexImageExpression =(lvalue ComplexImageExpression, ComplexImageExpression)

¬†

RealImageExpression =(lvalue RealImageExpression, RealImageExpression)

¬†

ImageExpression =(lvalue ImageExpression, ImageExpression)

¬†

ImageExpression =(lvalue BasicImage, ImageExpression)

¬†

RGBNumberExpression =(lvalue RGBNumberExpression *dst*, RGBNumberExpression *src*)

Assign src to dst.

¬†

ComplexNumberExpression =(lvalue ComplexNumberExpression *dst*, ComplexNumberExpression *src*)

Assign src to dst.

¬†

Number =(NumberVariable *dst*, Number *src*)

¬†

RealNumberExpression =(lvalue RealNumberExpression *dst*, RealNumberExpression *src*)

Assign src to dst.

¬†

LinePlotImageDisplay =(LinePlotImageDisplay *dst*, LinePlotImageDisplay *lpid*)

¬†

LinePlotImageDisplay =(LinePlotImageDisplay *dst*, !)

¬†

String =(String *dst*, !)

Assigns NULL to dst string.

¬†

String =(String *dst*, String *src*)

Assigns src string to dst string.

¬†

Throwable =(Throwable *dst*, !)

Assigns NULL to dst throwable.

¬†

Throwable =(Throwable *dst*, Throwable *src*)

Assigns src throwable to dst throwable.

¬†

`Function =( Function dst, ! )`

Assigns NULL to dst function.

¬†

`Function =( Function dst, Function src )`

Assigns src function to dst function.

¬†

`ScriptObject =( ScriptObject scriptObject1, ! )`

¬†

`ScriptObject =( ScriptObject scriptObject1, ScriptObject scriptObject2 )`

¬†

`ROI =( ROI, ! )`

Assign a region of interest to a new variable.

¬†

`ROI =( ROI, ROI )`

Assign a region of interest to a new variable.

¬†

`RasterImageDisplay =( RasterImageDisplay dst, ! )`

¬†

`RasterImageDisplay =( RasterImageDisplay dst, RasterImageDisplay rid )`

¬†

`TagGroup =( TagGroup tagGroup1, ! )`

¬†

`TagGroup =( TagGroup tagGroup1, TagGroup tagGroup2 )`

¬†

`LinePlotImageDisplay =( LinePlotImageDisplay dst, ImageDisplay id )`

¬†

`RasterImageDisplay =( RasterImageDisplay dst, ImageDisplay id )`

¬†

`RealNumberExpression ==( RGBNumberExpression, RGBNumberExpression )`

Return 1 if the left RGB value is equal to the right RGB value and 0 otherwise.

¬†

`RealNumberExpression ==( ComplexNumberExpression, ComplexNumberExpression )`

Return 1 if the left number is equal to the right number and 0 otherwise.

¬†

`Boolean ==( Number v1, Number v2 )`

¬†

`RealNumberExpression ==( RealNumberExpression, RealNumberExpression )`

Return 1 if the left number is equal to the right number and 0 otherwise.

¬†

`Boolean ==( String, String )`

Returns true if the two strings are equal character-by-character.

¬†

`RealNumberExpression >( RealNumberExpression, RealNumberExpression )`

Return 1 if the left number is greater than the right number and 0 otherwise.

¬†

`RealNumberExpression >=( RealNumberExpression, RealNumberExpression )`

Return 1 if the left number is greater than or equal to the right number and 0 otherwise.

¬†

`RealNumberExpression ?( RealNumberExpression condition, RealNumberExpression truenumber, RealNumberExpression falsenumber )`

Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.

¬†

`Number ?( Boolean cond, Number trueval, Number falseval )`

¬†

`ComplexNumberExpression ?( RealNumberExpression condition, ComplexNumberExpression truenumber, ComplexNumberExpression falsenumber )`

Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.

¬†

`RGBNumberExpression ?( RealNumberExpression condition, RGBNumberExpression truenumber, RGBNumberExpression falsenumber )`

Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.

¬†

`Number [( String, Number index )`

Returns the numeric value in the string's encoding of the first character of the string.

¬†

```
RealImage [( RealImage )
```
¬†
```
ComplexImage [( ComplexImage )
```
¬†
```
RGBImage [( RGBImage )
```
¬†
```
BasicImage [( BasicImage, Number x, Number y )
```
Returns a pixel in the given real image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†
```
BasicImage [( BasicImage, Number x, Number y, Number z )
```
Returns a pixel in the given real image at the position [x,y,z]. This operator is equivalent to Index(image,x,y,z).
¬†
```
BasicImage [( BasicImage, Number top, Number left, Number bottom, Number right )
```
Returns the sub-area image specified by the coordinates [top,left,bottom,right].
¬†
```
BasicImage [( BasicImage, Number y1, Number x1, Number z1, Number y2, Number x2, Number z2 )
```
Returns the sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†
```
RealImage [( RealImage, Number x, Number y )
```
Returns a pixel in the given real image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†
```
RealImage [( RealImage, Number x, Number y, Number z )
```
Returns a pixel in the given real image at the position [x,y,z]. This operator is equivalent to Index(image,x,y,z).
¬†
```
RealImage [( RealImage, Number top, Number left, Number bottom, Number right )
```
Returns the sub-area image specified by the coordinates [top,left,bottom,right].
¬†
```
RealImage [( RealImage, Number y1, Number x1, Number z1, Number y2, Number x2, Number z2 )
```
Returns the sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†
```
RealImageExpression [( RealImage image, RealImageExpression x, RealImageExpression y )
```
¬†
```
RealImageExpression [( RealImage image, RealImageExpression x, RealImageExpression y,
RealImageExpression z )
```
¬†
```
ComplexImage [( ComplexImage, Number top, Number left, Number bottom, Number right )
```
Returns the complex sub-area image specified by the coordinates [top,left,bottom,right].
¬†
```
ComplexImage [( ComplexImage, Number y1, Number x1, Number z1, Number y2, Number x2, Number z2 )
```
Returns the complex sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†
```
ComplexImage [( ComplexImage, Number x, Number y )
```
Returns a complex pixel in the given complex image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†
```
ComplexImageExpression [( ComplexImage, RealImageExpression x, RealImageExpression y )
```
¬†
```
ComplexImageExpression [( ComplexImage image, RealImageExpression x, RealImageExpression y,
RealImageExpression z )
```
¬†
```
RGBImage [( RGBImage, Number x, Number y )
```
Returns a RGB pixel in the given RGB image at the position [x,y]. This operator is equivalent to Index(image,x,y).
¬†
```
RGBImage [( RGBImage, Number top, Number left, Number bottom, Number right )
```
Returns the RGB sub-area image specified by the coordinates [top,left,bottom,right].
¬†
```
RGBImage [( RGBImage, Number y1, Number x1, Number z1, Number y2, Number x2, Number z2 )
```
Returns the RGB sub-area image specified by the coordinates [x,y] where x and y are 3-D positions.
¬†
```
RGBImageExpression [( RGBImage, RealImageExpression x, RealImageExpression y )
```
¬†
```
RGBImageExpression [( RGBImage image, RealImageExpression x, RealImageExpression y,
RealImageExpression z )
```
¬†
```
Boolean ||( Boolean v1, Boolean v2 )
```
¬†
```
RealNumberExpression ||( RealNumberExpression, RealNumberExpression )
```

Return the logical OR of the two numbers.
¬†
`void` **AbortAcquisitionDaemon**`( ImageReference )`
Abort all acquisition daemons associated with the image.
¬†
`void` **AbortAllAcquisitionDaemons**`( )`
Abort all acquisition daemons with any image within the application.
¬†
`RealNumberExpression` **abs**`( ComplexNumberExpression )`
Return the absolute value of the modulus of a complex number.
¬†
`Number` **abs**`( Number` *`v1`* `)`
¬†
`RealNumberExpression` **abs**`( RealNumberExpression )`
Return the absolute value of the number.
¬†
`Number` **acos**`( Number` *`v1`* `)`
¬†
`RealNumberExpression` **acos**`( RealNumberExpression )`
Return the arc-cosine of the number.
¬†
`void` **AddArrayMaskToImage**`( ImageReference` *`mask`*`, Number` *`x_center`*`, Number` *`y_center`*`, Number` *`x_radius`*`, Number` *`y_radius`*`, Number` *`x_v1`*`, Number` *`y_v1`*`, Number` *`x_v2`*`, Number` *`y_v2`*`, Number` *`filter_length`*`, Boolean` *`do_inverse`* `)`
¬†
`void` **AddBandPassMaskToImage**`( ImageReference` *`mask`*`, Number` *`x_center`*`, Number` *`y_center`*`, Number` *`band_radius_1`*`, Number` *`band_radius_2`*`, Number` *`filter_length`*`, Boolean` *`do_inverse`* `)`
¬†
`Number` **AddDaemonHack**`( ImageReference, Number` *`table`* `)`
Add a daemon component for the image. See the SDK documentation for more information.
¬†
`void` **AddOvalMaskToImage**`( ImageReference` *`mask`*`, Number` *`t_oval`*`, Number` *`l_oval`*`, Number` *`b_oval`*`, Number` *`r_oval`*`, Number` *`filter_length`*`, Boolean` *`do_inverse`* `)`
¬†
`void` **AddPathToCopyToImageList**`( String` *`path`* `)`
Adds 'path' to the list of tags to transfer to acquired images.
¬†
`void` **AddScriptFileToMenu**`( String` *`fileName`*`, String` *`commandName`*`, String` *`menuName`*`, String` *`optionalSubMenuName`*`, Boolean` *`isLibrary`* `)`
Install the script in the file indicated by fileName into the application. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.
¬†
`void` **AddScriptFileToPackage**`( String` *`fileName`*`, String` *`packageName`*`, Number` *`packageLevel`*`, String` *`commandName`*`, String` *`menuName`*`, String` *`optionalSubMenuName`*`, Boolean` *`isLibrary`* `)`
Install the script in the file indicated by fileName into the package. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.
¬†
`void` **AddScriptToMenu**`( String` *`script`*`, String` *`commandName`*`, String` *`menuName`*`, String` *`optionalSubMenuName`*`, Boolean` *`isLibrary`* `)`
Install the script into the application. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.
¬†
`void` **AddScriptToPackage**`( String` *`script`*`, String` *`packageName`*`, Number` *`packageLevel`*`, String` *`commandName`*`, String` *`menuName`*`, String` *`optionalSubMenuName`*`, Boolean` *`isLibrary`* `)`
Install the script into the application. The commandName indicates the string by which this script will be known to the application. If the script is to be installed in the menu, the menuName and optionalSubMenuName parameters specify the menu. Pass 1 for isLibrary if the script is a library only and 0 to install it in the menu.
¬†
`void` **AddStringToList**`( ImageReference, String` *`tagPath`*`, String` *`string`* `)`
Adds the string to the image tag list indicated by tagPath.
¬†
`void` **AddStringToPersistentList**`( String` *`tagPath`*`, String` *`string`* `)`
Adds the string to the persistent tag list indicated by tagPath.

¬†

void **AddTagsToPackage**(TagGroup *tags*, String *packageName*, Number *packageLevel*, String *identifier*)

Install the tags into the package. The identifier is used to identify the tags in the packages. Clients should take care to use unique identifiers. See the Java model of naming classes.

¬†

void **AddTwinMaskToImage**(ImageReference *mask*, Number *x_center*, Number *y_center*, Number *x_offset*, Number *y_offset*, Number *x_radius*, Number *y_radius*, Number *filter_length*, Boolean *do_inverse*)

¬†

void **AddWedgeMaskToImage**(ImageReference *mask*, Number *x_center*, Number *y_center*, Number *x_v1*, Number *y_v1*, Number *x_v2*, Number *y_v2*, Number *filter_length*, Boolean *do_inverse*)

¬†

void **AdjustScriptMenuItem**(String *commandName*, String *menuName*, String *optionalSubMenuName*, String *newCommandName*, Boolean *enabled*, Boolean *checked*, Number *key*, Number *acceleratorPos*)

Adjusts the display characteristics of the given script menu item. NewCommandName specifies the new name for the menu item. The menu item will have to be referred to by that name from then forth. Enabled/checked indicate whether the item is enabled/checked. Key refers to the command key equivalent on the MacOS. Pass 0 to have no command key equivalent. AcceleratorPos refers to the position of the accelerator equivalent on Windows. Pass -1 to have to accelerator.

¬†

RealNumberExpression **AiryAi**(Number)

Return the Airy Ai function of the number.

¬†

RealNumberExpression **AiryBi**(Number)

Return the Airy Bi function of the number.

¬†

ScriptObject **alloc**(String *class_name*)

¬†

RealNumberExpression **alpha**(RGBNumberExpression)

Return the alpha portion of an RGB number.

¬†

Number **AnnotationType**(ImageReference, Number *annotationID*)

Return the annotation type indicated by the annotation ID within the image.

¬†

void **ApplicationGetBounds**(NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)

Gets the bounds of the main area of the application in application coordinates.

¬†

void **ApplicationGetOrigin**(NumberVariable *x*, NumberVariable *y*)

Gets the origin of the application in global coordinates.

¬†

void **ApplyDataBar**(ImageDisplay *imgDisp*)

Applies a data bar to the image.

¬†

Number **asc**(String, Number *index*)

Returns numeric value of he 'index'th character in of the string.

¬†

Number **asc**(String)

Returns the numeric value in ascii of the first character of the string.

¬†

Number **asin**(Number *v1*)

¬†

RealNumberExpression **asin**(RealNumberExpression)

Return the arc-sine of the number.

¬†

Number **atan**(Number *v1*)

¬†

RealNumberExpression **atan**(RealNumberExpression)

Return the arc-tangent of the number.

¬†

RealNumberExpression **atan2**(RealNumberExpression *y*, RealNumberExpression *x*)

Return the arc-tangent of the ratio of y/x.

¬†

Number **atan2**(Number *x*, Number *y*)

¬†

RealNumberExpression **atanh**(RealNumberExpression)

Return the hyperbolic arc-tangent of the number.

¬†

`RealImage` **`AutoCorrelate`**`( RealImage` *`source`*`)`
Return an image which is the result of the auto correlation of source.
¬†

`RealImage` **`AutoCorrelation`**`( RealImage` *`source`*`)`
Return an image which is the result of the auto correlation of source.
¬†

`Number` **`average`**`( RealImageExpression` *`image`*`)`
Return the average pixel value of the image expression.
¬†

`String` **`BaseN`**`( Number` *`n`*`, Number` *`base`*`)`
Returns the number as a base n string.
¬†

`String` **`BaseN`**`( Number` *`n`*`, Number` *`base`*`, Number` *`length`*`)`
Returns the number as a base n string of the given length.
¬†

`void` **`Beep`**`( )`
Play the current system beep.
¬†

`RealNumberExpression` **`BesselI`**`( Number, Number )`
Return the general Bessel I function of two numbers.
¬†

`RealNumberExpression` **`BesselJ`**`( Number, Number )`
Return the general Bessel J function of two numbers.
¬†

`RealNumberExpression` **`BesselK`**`( Number, Number )`
Return the general Bessel K function of two numbers.
¬†

`RealNumberExpression` **`BesselY`**`( Number, Number )`
Return the general Bessel Y function of two numbers.
¬†

`RealNumberExpression` **`Beta`**`( Number, Number )`
Return the beta function of two numbers.
¬†

`String` **`Binary`**`( Number` *`n`*`, Number` *`length`*`)`
Returns the number as a binary string of the given length.
¬†

`String` **`Binary`**`( Number` *`n`*`)`
Returns the number as a binary string.
¬†

`RealImage` **`BinaryImage`**`( String` *`title`*`, Number` *`d0`*`)`
Creates a 1D binary image of size [d0] with the given title.
¬†

`RealImage` **`BinaryImage`**`( String` *`title`*`, Number` *`d0`*`, Number` *`d1`*`)`
Creates a 2D binary image of size [d0,d1] with the given title.
¬†

`RealImage` **`BinaryImage`**`( String` *`title`*`, Number` *`d0`*`, Number` *`d1`*`, Number` *`d2`*`)`
Creates a 3D binary image of size [d0,d1,d2] with the given title.
¬†

`RealNumberExpression` **`BinomialCoefficient`**`( Number, Number )`
Return the binomial coefficient of two numbers.
¬†

`RealNumberExpression` **`BinomialRandom`**`( Number, Number )`
Return a random number with binomial distribution between [0,1)
¬†

`RealNumberExpression` **`blue`**`( RGBNumberExpression )`
Return the blue portion of an RGB number.
¬†

`void` **`break`**`( )`
¬†

`void` **`BrowseTagFile`**`( )`
Present an open file dialog to the user, allow them to select a tag file, and then allow them to browse through it.
¬†

`RealNumberExpression` **`Ceil`**`( RealNumberExpression )`
Return the number truncated to an integer (rounding towards positive infinity).

¬†
Number **ceil**(Number *v1*)
¬†
void **CheckHeap**( )
Checks the integrity of the application memory.
¬†
Boolean **ChooseMenuItem**(String *menu*, String *subMenu*, String *item*)
Choose the given menu item.
¬†
String **chr**(Number *n*)
Returns the ASCII character specified by n as a string.
¬†
ComplexNumberExpression **cis**(RealNumberExpression)
Return complex(cos(n),sin(n)) as a complex number, where n is a real number. This is a unit vector in the complex plane.
¬†
Function **ClassAddMethod**(String *class_name*, Function *method*)
¬†
ScriptObject **ClassNewObject**(String *class_name*)
¬†
void **ClassRemoveMethod**(String *class_name*, Function *method*)
¬†
void **CleanImage**(ImageReference)
Mark the image as having been saved.
¬†
void **ClearImage**(ImageReference)
Set each pixel in the image to zero.
¬†
void **ClearSelection**(ImageReference)
Remove selection (if any) from the image.
¬†
RealNumberExpression **clip**(RealNumberExpression *value*, RealNumberExpression *minimum*, RealNumberExpression *maximum*)
Return the value clipped to be in the range bounded by minimum and maximum.
¬†
Number **clip**(Number *val*, Number *min_*, Number *max_*)
¬†
Boolean **ClipboardGetAsPicture**(NumberVariable *picture*)
Gets the contents of the clipboard as a picture, if possible, and returns true if successful.
¬†
Boolean **ClipboardGetAsString**(String *str*)
Gets the contents of the clipboard as a string with the given encoding, if possible, and returns true if successful.
¬†
Boolean **ClipboardGetAsTagGroup**(TagGroup *tagGroup*)
Gets the contents of the clipboard as a tag group, if possible, and returns true if successful.
¬†
void **ClipboardSetAsPicture**(Number *picture*)
Sets the contents of the clipboard to the picture.
¬†
void **ClipboardSetAsString**(String)
Sets the contents of the clipboard to the text.
¬†
void **ClipboardSetAsTagGroup**(TagGroup *tagGroup*)
Sets the contents of the clipboard to the tag group.
¬†
void **CloseFile**(Number *file*)
Close the file. This function should be called to close a file whenever a file is opened.
¬†
void **CloseImage**(ImageReference)
Attempt to close the image. If the data has changed, a dialog box appears to ask the user to save the image before closing it.
¬†
void **CloseProgressWindow**( )
Close the progress window if it is open.
¬†
void **CloseTimeBar**( )
Closes the time bar.

¬†

Boolean **CommandDown**( )

Returns 1 if the command key is down and 0 otherwise.

¬†

ComplexNumberExpression **complex**(RealNumberExpression *real*, RealNumberExpression *imaginary*)

Returns a complex number with the value of real + i * imaginary.

¬†

ComplexImage **ComplexImage**(String *title*, Number *bytes*, Number *d0*)

Creates a 1D complex image of size [d0] with the given title. The bytes parameter can be 8 or 16 for single and double precision floating point numbers.

¬†

ComplexImage **ComplexImage**(String *title*, Number *bytes*, Number *d0*, Number *d1*)

Creates a 2D complex image of size [d0,d1] with the given title. The bytes parameter can be 8 or 16 for single and double precision floating point numbers.

¬†

ComplexImage **ComplexImage**(String *title*, Number *bytes*, Number *d0*, Number *d1*, Number *d2*)

Creates a 3D complex image of size [d0,d1,d2] with the given title. The bytes parameter can be 8 or 16 for single and double precision floating point numbers.

¬†

ComplexImage **ComplexToPacked**(ComplexImage *source*)

Creates a new packed complex image from the complex 8-byte source.

¬†

ComplexImage **ComplexToPacked**(ComplexImage *source*, Number *style*)

Creates a new packed complex image from the complex 8-byte source.

¬†

void **ComponentAddChildAfter**(Component *parent*, Component *child*, Component *annot_pos*)

Adds 'child' to 'parent's list of sub-annotations after 'annot_pos'.

¬†

void **ComponentAddChildAtBeginning**(Component *parent*, Component *child*)

Adds 'child' to the beginning of 'parent's list of sub-annotations.

¬†

void **ComponentAddChildAtEnd**(Component *parent*, Component *child*)

Adds 'child' to the end of 'parent's list of sub-annotations.

¬†

void **ComponentAddChildBefore**(Component *parent*, Component *child*, Component *annot_pos*)

Adds 'child' to 'parent's list of sub-annotations before 'annot_pos'.

¬†

Component **ComponentAddNewComponent**(Component *parent*, Number *type*, Number *f1*, Number *f2*, Number *f3*, Number *f4*)

Creates a new annotaiton of type 'type' and adds it to 'parent'

¬†

Component **ComponentClone**(Component *comp*, Boolean *doDeepCopy*)

Returns a identical copy of the component and all its sub-components, copying associated images if 'doDeepCopy' is true.

¬†

Number **ComponentCountChildren**(Component *comp*)

Returns the number of sub components.

¬†

Number **ComponentCountChildrenOfType**(Component *comp*, Number *type*)

Returns the number of sub-components of type 'type'.

¬†

void **ComponentGetBoundingRect**(Component *comp*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)

Gets the bounding rect of the annotation.

¬†

void **ComponentGetBoundingRectInView**(Component *comp*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)

Gets the bounding rect of the annotation.

¬†

Component **ComponentGetChild**(Component *comp*, Number *index*)

Returns the 'index'th sub-component of 'comp'.

¬†

Component **ComponentGetChildByID**(Component *comp*, Number *ID*)

Returns the component child of 'comp' with id 'ID'.

¬†

void **ComponentGetChildToLocalTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*,

NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from child to local coordinates.
¬†
void **ComponentGetChildToPageTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from child to page coordinates.
¬†
void **ComponentGetChildToViewTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from child to view coordinates.
¬†
void **ComponentGetChildToWindowTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from child to window coordinates.
¬†
Boolean **ComponentGetControlPoint**(Component *comp*, Number *loc*, NumberVariable *x*, NumberVariable *y*)
Returns the value '(x,y)' associated with the control point, and returns 'true' if the control point is valid
¬†
Component **ComponentGetDescendentByID**(Component *comp*, Number *ID*)
Returns the component child of 'comp' with id 'ID'.
¬†
Number **ComponentGetDrawingMode**(Component *comp*)
Gets the drawing mode of the image doucment component.
¬†
Number **ComponentGetFillMode**(Component *comp*)
Gets the fill mode of the image doucment component.
¬†
Number **ComponentGetFontAttributes**(Component *comp*)
Gets the attributes of the component's font.
¬†
String **ComponentGetFontFaceName**(Component *comp*)
Gets the face name of the component's font.
¬†
void **ComponentGetFontInfo**(Component *comp*, String *faceName*, NumberVariable *attributes*, NumberVariable *size*, NumberVariable *text_encoding*)
Gets a description of the component's font.
¬†
void **ComponentGetFontInfo**(Component *comp*, String *faceName*, NumberVariable *attributes*, NumberVariable *size*)
Gets a description of the component's font.
¬†
Number **ComponentGetFontSize**(Component *comp*)
Gets the point size of the component's font.
¬†
Number **ComponentGetID**(Component *annot*)
Gets the unique identifier of the annotation in the image document.
¬†
ImageDocument **ComponentGetImageDocument**(Component *annot*)
Gets the image document associated with the annotation.
¬†
void **ComponentGetLocalToPageTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from local to page coordinates.
¬†
void **ComponentGetLocalToViewTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from local to view coordinates.
¬†
void **ComponentGetLocalToWindowTransform**(Component *comp*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Gets the transformation from local to window coordinates.
¬†
Component **ComponentGetNthChildOfType**(Component *comp*, Number *type*, Number *index*)
Returns the nth sub-component of type 'type'.
¬†
Component **ComponentGetParentComponent**(Component *comp*)

Gets the parent component of 'comp', if any.
¬†
`ImageDisplay` **`ComponentGetParentImageDisplay`**`( Component` *`comp`* `)`
Gets the parent image display of the 'comp', if any.
¬†
`void` **`ComponentGetRect`**`( Component` *`comp`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the rectangle of the annotation.
¬†
`void` **`ComponentGetRectInView`**`( Component` *`comp`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Gets the rectangle of the annotation.
¬†
`TagGroup` **`ComponentGetTagGroup`**`( Component` *`annot`* `)`
Gets the tag group associated with the annotation.
¬†
`Number` **`ComponentGetType`**`( Component` *`annot`* `)`
Gets the type of the annotation.
¬†
`Boolean` **`ComponentIsOfType`**`( Component` *`annot`*`, Number` *`type`* `)`
Gets the type of the annotation.
¬†
`Boolean` **`ComponentIsSelected`**`( Component` *`comp`* `)`
Returns whether the component is selected.
¬†
`Boolean` **`ComponentIsValid`**`( Component` *`annot`* `)`
Returns true if 'annot' points to a valid object.
¬†
`Component` **`ComponentNullify`**`( ! )`
¬†
`void` **`ComponentOffsetControlPoint`**`( Component` *`comp`*`, Number` *`loc`*`, Number` *`x`*`, Number` *`y`*`, Number` *`restrict_style`* `)`
Changes the control point 'loc' of 'comp' by '(x,y)' using restrictions specified by 'restrict_style'.
¬†
`void` **`ComponentPositionAroundPoint`**`( Component` *`comp`*`, Number` *`new_x`*`, Number` *`new_y`*`, Number` *`rel_x`*`, Number` *`rel_y`*`, Boolean` *`horz`*`, Boolean` *`vert`* `)`
Moves the annotation so if 'horz', the 'rel_x' horizontal point in the bounding rect is at 'new_x', and if 'vert', the 'rel_y' vertical point in the bounding rect is at 'new_y'
¬†
`void` **`ComponentRemoveFromParent`**`( Component` *`comp`* `)`
Removes the image document component from its parent.
¬†
`void` **`ComponentSetControlPoint`**`( Component` *`comp`*`, Number` *`loc`*`, Number` *`x`*`, Number` *`y`*`, Number` *`restrict_style`* `)`
Sets the control point 'loc' of 'comp' to '(x,y)' using restrictions specified by 'restrict_style'.
¬†
`void` **`ComponentSetDrawingMode`**`( Component` *`comp`*`, Number` *`mode`* `)`
Sets the drawing mode of the image document component.
¬†
`void` **`ComponentSetFillMode`**`( Component` *`comp`*`, Number` *`mode`* `)`
Sets the fill mode of the image document component.
¬†
`void` **`ComponentSetFontAttributes`**`( Component` *`comp`*`, Number` *`attributes`* `)`
Sets the attributes of the component's font.
¬†
`void` **`ComponentSetFontFaceName`**`( Component` *`comp`*`, String` *`face_name`* `)`
Sets the face name of the component's font.
¬†
`void` **`ComponentSetFontInfo`**`( Component` *`comp`*`, String` *`face_name`*`, Number` *`attributes`*`, Number` *`size`*`, Number` *`text_encoding`* `)`
Sets the font information of the component's font.
¬†
`void` **`ComponentSetFontInfo`**`( Component` *`comp`*`, String` *`face_name`*`, Number` *`attributes`*`, Number` *`size`* `)`
Sets the font information of the component's font.
¬†
`void` **`ComponentSetFontSize`**`( Component` *`comp`*`, Number` *`size`* `)`

Sets the point size of the component's font.
¬†
void **ComponentSetRect**(Component *comp*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the rectangle of the annotation.
¬†
void **ComponentSetSelected**(Component *comp*, Boolean *select*)
Sets the selection status of the component.
¬†
void **ComponentTransformCoordinates**(Component *comp*, Number *off_x*, Number *off_y*, Number *scale_x*, Number *scale_y*)
Transforms the component by the specified transform.
¬†
ComplexNumberExpression **conjugate**(ComplexNumberExpression)
Return the conjugate of the complex number.
¬†
void **ConnectObject**(Number *object*, String *message*, String *ident*, ScriptObject *scriptObject*, String *method*)
Build the connection with the given object and message under ident.
¬†
void **continue**( )
¬†
Boolean **ContinueCancelDialog**(String *prompt*)
Puts up a dialog with both a Continue button and Cancel button. Returns 1 for Continue and 0 for Cancel.
¬†
Boolean **ControlDown**( )
Returns 1 if the control key is down and 0 otherwise.
¬†
ImageDisplay **Convert_Component_ImageDisplay**(Component *annot*)
¬†
LinePlotImageDisplay **Convert_ImageDisplay_LinePlotImageDisplay**(ImageDisplay *imgDisp*)
¬†
RasterImageDisplay **Convert_ImageDisplay_RasterImageDisplay**(ImageDisplay *imgDisp*)
¬†
SurfacePlotImageDisplay **Convert_ImageDisplay_SurfacePlotImageDisplay**(ImageDisplay *imgDisp*)
¬†
ComplexImage **ConvertBiCi**(BasicImage *bi*)
¬†
ComplexImageExpression **ConvertBiCie**(BasicImage)
¬†
ComplexImageExpression **ConvertBiCieLValue**(BasicImage)
¬†
RGBImage **ConvertBiGi**(BasicImage *bi*)
¬†
RGBImageExpression **ConvertBiGie**(BasicImage)
¬†
RGBImageExpression **ConvertBiGieLValue**(BasicImage)
¬†
ImageExpression **ConvertBiNie**(BasicImage)
¬†
ImageExpression **ConvertBiNieLValue**(BasicImage)
¬†
RealImage **ConvertBiRi**(BasicImage *bi*)
¬†
RealImageExpression **ConvertBiRie**(BasicImage)
¬†
RealImageExpression **ConvertBiRieLValue**(BasicImage)
¬†
ComplexImageExpression **ConvertCiCie**(ComplexImage)
¬†
ComplexImageExpression **ConvertCiCieLValue**(ComplexImage)
¬†
ComplexImage **ConvertCieCi**(ComplexImageExpression)
¬†
ComplexNumberExpression **ConvertCnCne**(ComplexNumber)
¬†
ComplexNumberExpression **ConvertCnCneLValue**(ComplexNumber)

¬†
ComplexImageExpression **ConvertCneCie**(ComplexNumberExpression)
¬†
ComplexNumber **ConvertCneCn**(ComplexNumberExpression)
¬†
String **ConvertEndOfLine**(Number *eol_format*, String *str*)
Converts the string to the end-of-line format indicated by 'eol_format_index': 0 == platform format, 1 = Macintosh format, 2 = Windows format.
¬†
RGBImage **ConvertGieGi**(RGBImageExpression)
¬†
RGBImageExpression **ConvertGiGie**(RGBImage)
¬†
RGBImageExpression **ConvertGiGieLValue**(RGBImage)
¬†
RGBImageExpression **ConvertGneGie**(RGBNumberExpression)
¬†
RGBNumber **ConvertGneGn**(RGBNumberExpression)
¬†
RGBNumberExpression **ConvertGnGne**(RGBNumber)
¬†
RGBNumberExpression **ConvertGnGneLValue**(RGBNumber)
¬†
void **ConvertImageData**(ImageReference *from*, ImageReference *to*)
¬†
void **ConvertImageDataSlice_2D**(ImageReference *from*, ImageReference *to*, Number *x_1*, Number *y_1*, Number *f_x_0*, Number *f_y_0*, Number *f_x_d*, Number *f_x_s*, Number *f_y_d*, Number *f_y_s*, Number *t_x_0*, Number *t_y_0*, Number *t_x_d*, Number *t_x_s*, Number *t_y_d*, Number *t_y_s*)
¬†
BasicImage **ConvertNieNi**(ImageExpression)
¬†
ComplexImageExpression **ConvertRieCie**(RealImageExpression)
¬†
RGBImageExpression **ConvertRieGie**(RealImageExpression)
¬†
RealImage **ConvertRieRi**(RealImageExpression)
¬†
RealImageExpression **ConvertRiRie**(RealImage)
¬†
RealImageExpression **ConvertRiRieLValue**(RealImage)
¬†
ComplexNumberExpression **ConvertRneCne**(RealNumberExpression)
¬†
RGBNumberExpression **ConvertRneGne**(RealNumberExpression)
¬†
RealImageExpression **ConvertRneRie**(RealNumberExpression)
¬†
RealNumber **ConvertRneRn**(RealNumberExpression)
¬†
RealNumberExpression **ConvertRnRne**(RealNumber *sexp*)
¬†
RealNumberExpression **ConvertRnRneLValue**(RealNumber)
¬†
void **ConvertToByte**(ImageReference)
Converts the given image to unsigned integer 1-byte data.
¬†
void **ConvertToComplex**(ImageReference)
Converts the given image to complex single precision data.
¬†
void **ConvertToFloat**(ImageReference)
Converts the given image to single precision real data.
¬†
void **ConvertToLong**(ImageReference)
Converts the given image to signed integer 4-byte data.
¬†
void **ConvertToPackedComplex**(ImageReference)

Converts the given image to packed complex data.
¬†
`void` **`ConvertToShort`**`( ImageReference )`
Converts the given image to signed integer 2-byte data.
¬†
`void` **`ConvertXieAny`**`( ImageExpression )`
¬†
`void` **`ConvertXneAny`**`( NumberExpression )`
¬†
`RealImage` **`Convolution`**`( ImageReference` *`source`*`, ImageReference` *`kernel`*` )`
Creates a new image that is the convolution of the source image with the kernel. The kernel should be less than 7x7.
¬†
`ComplexNumberExpression` **`cos`**`( ComplexNumberExpression )`
Return the cosine of the number
¬†
`Number` **`cos`**`( Number` *`v1`*` )`
¬†
`RealNumberExpression` **`cos`**`( RealNumberExpression )`
Return the cosine of the number.
¬†
`ComplexNumberExpression` **`cosh`**`( ComplexNumberExpression )`
Return the hyperbolic cosine of the number.
¬†
`RealNumberExpression` **`cosh`**`( RealNumberExpression )`
Return the hyperbolic cosine of the number.
¬†
`Number` **`cosh`**`( Number` *`v1`*` )`
¬†
`Number` **`CountAllImages`**`( )`
Returns the number of images.
¬†
`Number` **`CountAnnotations`**`( ImageReference )`
Return the number of annotations within the image.
¬†
`Number` **`CountDocumentWindows`**`( )`
Returns the number of document windows.
¬†
`Number` **`CountDocumentWindowsOfType`**`( Number` *`type`*` )`
Returns the number of document windows with type 'type'.
¬†
`Number` **`CountFloatingWindows`**`( )`
Returns the number of floating windows.
¬†
`Number` **`CountImageDocuments`**`( )`
Returns the number of image documents.
¬†
`Number` **`CountImages`**`( )`
Count the number of images.
¬†
`Number` **`CountListEntries`**`( ImageReference, String` *`tagPath`*` )`
Returns the number of tags within the given image tag list or group indicated by tagPath.
¬†
`Number` **`CountPersistentListEntries`**`( String` *`tagPath`*` )`
Returns the number of tags within the given persistent tag list or group indicated by tagPath.
¬†
`Number` **`CountScreens`**`( )`
Returns the number of screens.
¬†
`Number` **`CountScriptFunctions`**`( String` *`fnName`*` )`
Returns the number of script functions having name 'fnName'.
¬†
`Number` **`CreateArrowAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*
`)`
Create an arrow annotation in the image with the endpoints [top,left] and [bottom,right]. Return the new annotation's ID.
¬†

Number **CreateBoxAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Create a box annotation in the image with the coordinates [top, left, bottom, right]. Return the new annotation's ID.
¬†

RealImage **CreateByteImage**(String *title*, Number *width*, Number *height*)
Creates a 2D unsigned 1-byte integer image of size [width,height] with the given title.
¬†

ComplexImage **CreateComplexImage**(String *title*, Number *width*, Number *height*)
Creates a 2D single precision complex image of size [width,height] with the given title.
¬†

void **CreateDirectory**(String *fileName*)
Create a folder named fileName.
¬†

Number **CreateDoubleArrowAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Create an double ended arrow annotation in the image with the endpoints [top,left] and [bottom,right]. Return the new annotation's ID.
¬†

void **CreateFile**(String *fileName*)
Create a file named fileName.
¬†

Number **CreateFileForWriting**(String *fileName*)
Create and open the file for writing. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.
¬†

RealImage **CreateFloatImage**(String *title*, Number *width*, Number *height*)
Creates a 2D single precision float image of size [width,height] with the given title.
¬†

ImageDocument **CreateImageDocument**(String *title*)
Creates an empty image document.
¬†

RGBImage **CreateImageFromDisplay**(ImageReference)
Convert the display of image to an RGB image.
¬†

RealImage **CreateInlineArray**(RealImage, RealNumber...)
¬†

ComplexImage **CreateInlineArray**(ComplexImage, ComplexNumber...)
¬†

RGBImage **CreateInlineArray**(RGBImage, RGBNumber...)
¬†

void **CreateJavaInterface**( )
¬†

void **CreateJavaStubs**( )
¬†

Number **CreateLineAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Create an line annotation in the image with the endpoints [top,left] and [bottom,right]. Return the new annotation's ID.
¬†

RealImage **CreateLongImage**(String *title*, Number *width*, Number *height*)
Creates a 2D signed 4-byte integer image of size [width,height] with the given title.
¬†

ImageReference **CreateMaskFromAnnotations**(RasterImageDisplay *rid*, Number *filter_length*, Boolean *is_opaque*, NumberVariable *has_mask*)
¬†

void **CreateOldPlugInInterfaceFiles**(Number *os_id*)
¬†

Number **CreateOvalAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)
Create an oval annotation in the image with the coordinates [top, left, bottom, right]. Return the new annotation's ID.
¬†

ComplexImage **CreatePackedComplexImage**(String *title*, Number *width*, Number *height*)
Creates a 2D packed complex image of size [width,height] with the given title.
¬†

Number **CreatePictureAnnotation**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*, ! *PicHandle*)
Create a picture annotation in the image with the coordinates [top, left, bottom, right] from the picHandle. Return the new annotation's ID.
¬†

void **CreatePlugInInterfaceFiles**(Number *os_id*)
¬†

RGBImage **CreateRGBImage**(String *title*, Number *width*, Number *height*)

Creates a 2D RGB image of size [width,height] with the given title.
¬†
`RGBImage` **`CreateRGBImageFromPicture`**`( Number` *`picture`* `)`
Create an RGB image by drawing into it with a picture
¬†
`ROI` **`CreateROI`**`( )`
Creates an empty region of interest.
¬†
`RealImage` **`CreateShortImage`**`( String` *`title`*`, Number` *`width`*`, Number` *`height`* `)`
Creates a 2D signed 2-byte integer image of size [width,height] with the given title.
¬†
`void` **`CreateShortScriptHTMLDocumentation`**`( String` *`directory`* `)`
¬†
`Number` **`CreateTextAnnotation`**`( ImageReference, Number` *`top`*`, Number` *`left`*`, String` *`text`* `)`
Create a text annotation in the image at the position [top,left] with text. Return the new annotation's ID.
¬†
`void` **`CreateXMLFunctionDescriptions`**`( )`
¬†
`RealImage` **`CrossCorrelate`**`( RealImage` *`source1`*`, RealImage` *`source2`* `)`
Return an image which is the result of the cross correlation of source1 and source2.
¬†
`RealImage` **`CrossCorrelation`**`( RealImage` *`source1`*`, RealImage` *`source2`* `)`
Return an image which is the result of the cross correlation of source1 and source2.
¬†
`RealImage` **`CrossProduct`**`( RealImage` *`a`*`, RealImage` *`b`* `)`
Return the matrix cross product image of matrix images a and b.
¬†
`String` **`DateStamp`**`( )`
Return a string representing the current date and time.
¬†
`String` **`Decimal`**`( Number` *`n`* `)`
Returns the number as a decimal string.
¬†
`String` **`Decimal`**`( Number` *`n`*`, Number` *`length`* `)`
Returns the number as a decimal string of the given length.
¬†
`void` **`Delay`**`( Number )`
Delay for the given number of tick counts.
¬†
`void` **`DeleteAnnotation`**`( ImageReference, Number` *`annotationID`* `)`
Delete the annotation indicated by the annotationID within the image.
¬†
`void` **`DeleteAnnotationNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`* `)`
Removes the annotation note with the label noteLabel.
¬†
`void` **`DeleteDirectory`**`( String` *`dirName`* `)`
Deletes the folder named dirName.
¬†
`void` **`DeleteFile`**`( String` *`fileName`* `)`
Delete the file.
¬†
`void` **`DeleteImage`**`( ImageVariable )`
Close the image without asking the user to save it.
¬†
`void` **`DeleteImageFile`**`( String` *`fileName`* `)`
Delete the image file.
¬†
`void` **`DeleteListEntry`**`( ImageReference, String` *`tagPath`*`, Number` *`index`* `)`
Deletes the tag with the given index from within the image tag list indicated by tagPath.
¬†
`void` **`DeleteNote`**`( ImageReference, String` *`noteLabel`* `)`
Removes the image note with the label noteLabel.
¬†
`void` **`DeletePersistentListEntry`**`( String` *`tagPath`*`, Number` *`index`* `)`
Deletes the tag with the given index from within the persistent tag list indicated by tagPath.

¬†
void **DeletePersistentNote**(String *noteLabel*)
Removes the note with the label noteLabel from the persistent note list.
¬†
void **DeselectAnnotation**(ImageReference, Number *annotationID*)
Deselect the annotation indicated by the annotationID within the image.
¬†
void **DestroyPicture**(NumberVariable *picture*)
Destroy a picture
¬†
void **DisconnectObject**(Number *object*, String *message*, String *ident*)
Break the connection specified by ident for the given object and message.
¬†
void **DisplayAt**(ImageReference, Number *x*, Number *y*)
Display the image's image document if it is not display already and moves the window position to [x,y] screen coordinates.
¬†
RealNumberExpression **distance**(RealNumberExpression *x*, RealNumberExpression *y*)
Return sqrt(x * x + y * y).
¬†
Number **distance**(Number *v1*, Number *v2*)
¬†
void **DM_Utility_GetScriptMemoryUsage**( )
¬†
Boolean **DoesClassExist**(String *class_name*)
¬†
Boolean **DoesDirectoryExist**(String *dirName*)
Returns 'true' if the named directory exists
¬†
Boolean **DoesFileExist**(String *dirName*)
Returns 'true' if the named file exists
¬†
Boolean **DoesFunctionExist**(String *fnName*)
Determines if the given function exists.
¬†
Boolean **DoesImageExist**(Number *imageID*)
Determine if the image with imageID exists and returns 1 if it does; return 0 otherwise.
¬†
void **DoEvents**( )
Process all pending MacOS events.
¬†
void **DontInterpret**(Number *x*)
¬†
ComplexNumber **DotProduct**(ComplexImageExpression, ComplexImageExpression)
Return the dot product of two image expressions (which are treated as vectors).
¬†
Number **DotProduct**(RealImageExpression, RealImageExpression)
Return the dot product of two image expressions (which are treated as vectors).
¬†
void **EditorWindowAddText**(DocumentWindow *window*, String *text*)
Appends the text to a editor window.
¬†
void **EditorWindowGetFont**(DocumentWindow *window*, String *face_name*, NumberVariable *attributes*, NumberVariable *size*, NumberVariable *text_encoding*)
Gets the font of a script window.
¬†
String **EditorWindowGetText**(DocumentWindow *window*)
Gets the text in an editor window.
¬†
Boolean **EditorWindowPrint**(DocumentWindow *window*)
Prints the editor window.
¬†
void **EditorWindowSaveToFile**(DocumentWindow *window*, String *path*)
Saves the editor window to the specified path.
¬†
void **EditorWindowSetFont**(DocumentWindow *window*, String *face_name*, Number *attributes*, Number *size*,

Number *text_encoding*)
Sets the font of a script window.
¬†
void **EditorWindowSetText**(DocumentWindow *window*, String *text*)
Sets the text in an editor window.
¬†
void **EMBeamShift**(Number *xAmount*, Number *yAmount*)
Shift the beam by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMChangeFocus**(Number *amount*)
Change the focus by amount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMChangeStigmation**(Number *xAmount*, Number *yAmount*)
Change the stigmation by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMChangeTilt**(Number *xAmount*, Number *yAmount*)
Change the tilt by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMCloseCommunication**( )
Close communication to the microscope. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMImageShift**(Number *xAmount*, Number *yAmount*)
Shift the image by xAmount, yAmount. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareImageShift**( )
Prepare image shift. Call this before a sequence of image shift changes. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareShift**( )
Prepare beam shift. Call this before a sequence of beam shift changes. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareStigmation**( )
Prepare stigmation. Call this before a sequence of stigmation changes. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMPrepareTilt**( )
Prepare tilt. Call this before a sequence of tilt changes. The EM Control Plug-in must be currently installed and configured.
¬†
void **EMSetupCommunication**( )
Setup communication with the microscope. The EM Control Plug-in must be currently installed and configured.
¬†
RealNumberExpression **erf**(Number)
Return the error function of the number.
¬†
RealNumberExpression **erfc**(Number)
Return the complement of the error function of the number.
¬†
ComplexImageExpression **EvaluateXneEtc_CneFnToXieEtc_CieFn**(ComplexNumberExpression, ImageExpression...)
¬†
RGBImageExpression **EvaluateXneEtc_GneFnToXieEtc_GieFn**(RGBNumberExpression, ImageExpression...)
¬†
ImageExpression **EvaluateXneEtc_NneFnToXieEtc_NieFn**(NumberExpression, ImageExpression...)
¬†
RealImageExpression **EvaluateXneEtc_RneFnToXieEtc_RieFn**(RealNumberExpression, ImageExpression...)
¬†
void **ExecuteCompiledScripts**(Number *x*)
¬†
void **ExecutePersistentScriptGroup**(String *tagPath*, String *form*)
Execute a group of script functions in the persistent tag list indicated by tagPath. The actual scripts executed will be formed by sprintf'ing into the form parameter. The form parameter should contain exactly one %s in which will be inserted the function name.
¬†
Number **ExecuteScriptFile**(String *fileName*, Number *script_index*)
Executes the script file indicated by fileName and returns the exit value of that script. A script may specify a specific exit value by exiting with the exit(n) function. If a script does not use the exit() function the exit value will be 0.
¬†

Number **ExecuteScriptFile**(String *fileName*)
Executes the script file indicated by fileName and returns the exit value of that script. A script may specify a specific exit value by exiting with the exit(n) function. If a script does not use the exit() function the exit value will be 0.
¬†
void **ExecuteScriptGroup**(ImageReference *image*, String *tagPath*, String *form*)
Execute a group of script functions in the image tag list indicated by tagPath. The actual scripts executed will be formed by sprintf'ing into the form parameter. The form parameter should contain exactly one %s in which will be inserted the function name.
¬†
Number **ExecuteScriptString**(String *text*)
Executes the script text and returns the exit value of that script. A script may specify a specific exit value by exiting with the exit(n) function. If a script does not use the exit() function the exit value will be 0.
¬†
void **exit**(Number)
Stops execution of the current script immediately and returns the number as the result of the script.
¬†
Number **exp**(Number *v1*)
¬†
ComplexNumberExpression **exp**(ComplexNumberExpression)
Return the exponential of the number.
¬†
RealNumberExpression **exp**(RealNumberExpression)
Return the exponential of the number.
¬†
Number **exp1**(Number *v1*)
¬†
RealNumberExpression **exp1**(RealNumberExpression)
Return the exponential - 1 of the number.
¬†
Number **exp10**(Number *v1*)
¬†
RealNumberExpression **exp10**(RealNumberExpression)
Return 10 raised to the number.
¬†
Number **exp2**(Number *v1*)
¬†
RealNumberExpression **exp2**(RealNumberExpression)
Return 2 raised to the number.
¬†
RealNumberExpression **ExponentialRandom**( )
Return a random number with exponential distribution between [0,1).
¬†
ImageExpression **ExprSize**(Number *x*, Number *y*, Number *z*, ImageExpression *value*)
The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.
¬†
RGBImageExpression **ExprSize**(ImageReference *image*, RGBImageExpression *value*)
The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to the size of image expression and this function will return the value parameter.
¬†
RGBImageExpression **ExprSize**(Number *x*, Number *y*, Number *z*, RGBImageExpression *value*)
The ExprSize function is used to declare the size of a RGB image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.
¬†
ImageExpression **ExprSize**(Number *x*, Number *y*, ImageExpression *value*)
The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.
¬†
RGBImageExpression **ExprSize**(Number *x*, Number *y*, RGBImageExpression *value*)
The ExprSize function is used to declare the size of an RGB image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.
¬†
ComplexImageExpression **ExprSize**(Number *x*, Number *y*, Number *z*, ComplexImageExpression *value*)
The ExprSize function is used to declare the size of a complex image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.

¬†

`ComplexImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, ComplexImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a complex image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.

¬†

`ComplexImageExpression` **`ExprSize`**`( ImageReference` *`image`*`, ComplexImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of an image expression that would otherwise be of an undefined size. The size is set to the size of image expression and this function will return the value parameter.

¬†

`RealImageExpression` **`ExprSize`**`( ImageReference` *`image`*`, RealImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a real image expression that would otherwise be of an undefined size. The size is set to the size of image expression and this function will return the value parameter.

¬†

`RealImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, Number` *`z`*`, RealImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a real image expression that would otherwise be of an undefined size. The size is set to [x,y,z] and this function will return the value parameter.

¬†

`RealImageExpression` **`ExprSize`**`( Number` *`x`*`, Number` *`y`*`, RealImageExpression` *`value`* `)`

The ExprSize function is used to declare the size of a real image expression that would otherwise be of an undefined size. The size is set to [x,y] and this function will return the value parameter.

¬†

`void` **`Extract2D_Linear`**`( ImageReference` *`from`*`, ImageReference` *`to`*`, Number` *`extract_style`*`, Number` *`x_start`*`, Number` *`y_start`*`, Number` *`x_scale_0`*`, Number` *`y_scale_0`*`, Number` *`x_scale_1`*`, Number` *`y_scale_1`* `)`

¬†

`RealNumberExpression` **`Factorial`**`( Number )`

Return the factorial of the number.

¬†

`ComplexImage` **`FFT`**`( ComplexImage` *`source`* `)`

Creates a new complex 8-byte image from the FFT of the complex image source.

¬†

`ImageReference` **`FindFrontImage`**`( )`

Returns the front image. Doesn't throw exceptions.

¬†

`Function` **`FindFunctionBySignature`**`( String` *`signature`* `)`

Looks for a function that matches the given signature.

¬†

`ImageReference` **`FindImageByID`**`( Number` *`id`* `)`

Returns the image having the given id, or an invalid image of no image has that id.

¬†

`ImageReference` **`FindImageByIndex`**`( Number` *`index`* `)`

Returns the 'index'th image.

¬†

`ImageReference` **`FindImageByLabel`**`( String` *`label`* `)`

Returns the image having the given label, or an invalid image if no such image exists.

¬†

`ImageReference` **`FindImageByName`**`( String` *`name`* `)`

Returns the image having the given name, or an invalid image of no image has that name.

¬†

`ImageReference` **`FindNextImage`**`( ImageReference )`

Find the next image.

¬†

`ImageReference` **`FirstImage`**`( )`

Find the first image.

¬†

`void` **`FlipHorizontal`**`( ImageReference )`

Flips the image horizontally.

¬†

`void` **`FlipVertical`**`( ImageReference )`

Flips the image vertically.

¬†

`void` **`FloatingModelessDialog`**`( String` *`prompt`*`, String` *`buttonName`*`, Number` *`semaphore`* `)`

Present a floating window with the prompt and buttonName. When the user presses the button, the semaphore will be cleared. This function can only be used in the background.

¬†

Number **floor**(Number *v1*)
¬†
RealNumberExpression **Floor**(RealNumberExpression)
Return the number truncated to an integer (rounding towards negative infinity).
¬†
void **FM_ConjMultiplyPackedByPacked**(ImageReference *a*, ImageReference *b*)
Conjugate multiply packed image a by packed image b and store the result in b. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ConvertInt16ToFloat**(ImageReference *a*, ImageReference *b*)
Convert the signed 2-byte data in image a to real data and store the result in image b. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ConvertUInt8ToDisplay8**(ImageReference *src*, Number *top*, Number *left*, Number *bottom*, Number *right*, Number *dst*, Number *rowBytes*)
Copy the sub-area of unsigned 1-byte integer image src indicated by [top,left,bottom,right] to dst. The rowBytes parameter indicates the length of the row of dst. To copy to the screen, pass 0 for dst and rowBytes. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ConvertUInt8ToFloat**(ImageReference *a*, ImageReference *b*)
Convert the unsigned 1-byte data in image a to real data and store the result in image b. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_FinishDMA**(ImageReference *image*)
Finish an image for DMA access. This function should be called once for every FM_PrepareDMA() call.
¬†
void **FM_Flush**( )
Finish any pending operations on the array processor.
¬†
Number **FM_GetVarianceFloat**(ImageReference)
Return the variance of the image. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_ImageDataChanged**(ImageReference *image*)
Inform the application that the image has changed due to DMA or other memory access.
¬†
void **FM_MultiplyFloatByFloat**(ImageReference *a*, ImageReference *b*)
Multiply real image a by real image b and store the result in a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_MultiplyPackedByFloat**(ImageReference *a*, ImageReference *b*)
Multiply packed image a by real image b and store the result in a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_MultiplyPackedByScalar**(ImageReference *a*, Number)
Multiply packed image a by real and store the result in a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_PackedFFT**(ImageReference *a*)
Perform an in-place packed FFT on image a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_PackedIFFT**(ImageReference)
Perform an in-place packed inverse FFT on image a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_PackedLnModulusToImage**(ImageReference *imageSrc*, ImageReference *imageDst*, Number *lowLimit*, Number *highLimit*, Number *range*)
Store the unpacked log modulus of the packed complex image imageSrc into imageDst using the lowLimit, highLimit, and range parameters. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_PrepareDMA**(ImageReference *image*, NumberVariable *logicalAddress*, NumberVariable *physicalAddress*)
Prepare an image for DMA access by locking down the data and preventing it from being deleted. If DMA access is possible, store the physical address in the physicalAddress variable. Store the logical address in the logicalAddress variable.The FM_FinishDMA() function should be called once for every FM_PrepareDMA() call.
¬†

void **FM_SetMacOnly**(Boolean *b*)
Disable the plug-in array processor if present. This function may not be supported by plug-in array processors.
¬†
void **FM_ShiftCenterFloat**(ImageReference *a*)
Shift the center of image a in-place (after a forward FFT or before an inverse FFT). No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_SubtractMeanPacked**(ImageReference *a*)
Subtract the mean from the packed image a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_SubtractMultiply**(ImageReference *a*, ImageReference *b*, ImageReference *c*)
Subtract real image b from real image a, multiply the result by real image c, and store the result in image a. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_SurveyNormalPackedModulus**(ImageReference *image*, NumberVariable *lowLimit*, NumberVariable *highLimit*)
Survey packed image a and store the minimum and maximum values found into the lowLimit and highLimit variables. No data type checking is performed. This function uses the array processor if present.
¬†
void **FM_TurboFFT**(ImageReference *buffer*, ImageReference *fft*, ImageReference *dark*, ImageReference *gain*, Number *top*, Number *left*, Number *bottom*, Number *right*, Number *dst*, Number *rowBytes*)
Perform a turbo FFT on the real image fft minus the real dark image and multiplied by the real gain image. Store the resulting FFT into the packed complex image fft. Copy the sub-area of unsigned 1-byte integer image src indicated by [top,left,bottom,right] to dst. The rowBytes parameter indicates the length of the row of dst. To copy to the screen, pass 0 for dst and rowBytes. The real buffer image is a scratch area. No data type checking is performed. This function uses the array processor if present.
¬†
Boolean **FontManagerLookupFont**(String *base_name*, Number *text_encoding*, Number *kind*, String *face_name*, NumberVariable *size*)
Tries to match a font to the 'base_name', 'text_encoding', and 'kind' specified.
¬†
void **for**(RealNumber, Expression, Expression)
¬†
String **format**(Number, String *formatString*)
Convert the number to a string using the printf-style real format specified by formatString.
¬†
void **FormatDouble**(Number *x*, Number *err*, Number *nchars*, Number *use_num_sig*, Number *format*)
¬†
RealImageExpression **FourPointAverage**(RealImage *source*)
Returns an image expression that evaluates to half the size of the source expression. Each point in the return image expression is the average of the four corresponding points in the source expression.
¬†
ComplexImageExpression **FourPointAverage**(ComplexImage *source*)
Returns an image expression that evaluates to half the size of the source expression. Each point in the return image expression is the average of the four corresponding points in the source expression.
¬†
RGBImageExpression **FourPointAverage**(RGBImage *source*)
Returns an image expression that evaluates to half the size of the source expression. Each point in the return image expression is the average of the four corresponding points in the source expression.
¬†
Number **FPClassify**(Number *v*)
Returns the class of the floating point number.
¬†
void **FreeSemaphore**(Number)
Free the semaphore. Used only with background processing.
¬†
Number **FunctionCountParameters**(Function *func*)
Returns the number of parameters of the function.
¬†
String **FunctionGenerateStub**(Function *func*, Boolean *include_body*, Number *version*)
Generates a stub function in the format specified by version.
¬†
Boolean **FunctionIsValid**(Function *func*)
Returns true if 'func' is a valid object.
¬†
Function **FunctionNullify**(!)

Assigns NULL to dst function.
¬†
RealNumberExpression **Gamma**(Number)
Return the gamma of the number.
¬†
RealNumberExpression **GammaP**(Number, Number)
Return the incomplete gamma function of two numbers.
¬†
RealNumberExpression **GammaQ**(Number, Number)
Return the complement of the incomplete gamma function of two numbers.
¬†
RealNumberExpression **GammaRandom**(Number)
Return a random number with gamma distribution between [0,1).
¬†
RealNumberExpression **GaussianRandom**( )
Return a random number with gaussian distribution between [0,1).
¬†
void **Get1DSize**(ImageReference, NumberVariable *d0*)
Store the length of the 1D image into the d0 variable.
¬†
void **Get2DSize**(ImageReference, NumberVariable *d0*, NumberVariable *d1*)
Store the width and height of the 2D image into the d0 and d1 variables.
¬†
void **Get3DSize**(ImageReference, NumberVariable *d0*, NumberVariable *d1*, NumberVariable *d2*)
Store the x,y, and z sizes of the 3D image into the d0, d1, and d2 variables.
¬†
Boolean **GetAnnotationComplexNumberNote**(ImageReference, Number *annotationID*, String *noteLabel*, ComplexNumberVariable *noteValue*)
Stores the value of the annotation note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†
Boolean **GetAnnotationNumberNote**(ImageReference, Number *annotationID*, String *noteLabel*, NumberVariable *noteValue*)
Stores the value of the annotation note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†
void **GetAnnotationRect**(ImageReference, Number *annotationID*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Store the bounds of the annotation indicated by the annotationID within the image into the top, left, bottom, and right variables.
¬†
Boolean **GetAnnotationRectNote**(ImageReference, Number *annotationID*, String *noteLabel*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the value of the annotation note with label noteLabel into the top, left, bottom, and right variables. Returns 1 if the note exists returns 0 otherwise.
¬†
Boolean **GetAnnotationRGBNumberNote**(ImageReference, Number *annotationID*, String *noteLabel*, RGBNumberVariable *noteValue*)
Stores the value of the annotation note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†
Boolean **GetAnnotationStringNote**(ImageReference, Number *annotationID*, String *noteLabel*, String *noteValue*)
Copies the value of the annotation note with the label noteLabel into the noteBuffer variable. Returns 1 if the note exists returns 0 otherwise.
¬†
String **GetAnnotationStringNote**(ImageReference, Number *annotationID*, String *noteLabel*)
Returns the value of the annotation note with the label noteLabel. Returns an empty string if the note does not exist.
¬†
String **GetApplicationDirectory**(Number *index*, Boolean *create_if_necessary*)
Return one of the application directories. 0=current directory, 1=executable directory.
¬†
void **GetApplicationFont**(Number *which_font*, String *face_name*, NumberVariable *attributes*, NumberVariable *size*, NumberVariable *text_encoding*)
Gets the face name, attributes, size, and text encoding of the selected appliction font.
¬†
Boolean **GetApplicationInfo**(Number *info_kind*, NumberVariable *info*)
Get application info. For 'info_kind == 0', bit 2 of '*info' is set for demo versions.
¬†
DocumentWindow **GetApplicationWindow**( )

Gets the application window.
¬†
Boolean **GetBoolean**(String *prompt*, Boolean *initalValue*, NumberVariable *result*)
Puts up a dialog with the given prompt and allows the user to enter Boolean. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.
¬†
Boolean **GetCalibrationDialog**(Number *aw*, Number *ah*, NumberVariable *xs*, NumberVariable *ys*, String *initialUnitString*, String *unitString*)
Present the calibration dialog to the user. The calibrating pixel dimensions are passed as the [aw,ah] parameters. The resulting calibration is stored into the [xs,ys] parameters. The initial unit string is passed in and the resulting unit string is stored into the unitString variable.Returns 1 for OK and 0 for Cancel.
¬†
RGBImage **GetCLUT**(ImageReference)
Return the image's CLUT as a 256x1 RGB image.
¬†
Boolean **GetComplexNumberNote**(ImageReference, String *noteLabel*, ComplexNumberVariable *noteValue*)
Stores the value of the image note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†
Number **GetDaemonComponent**(ImageReference)
Return the daemon component for the image. See the SDK documentation for more information.
¬†
String **GetDate**(Number *dateFormat*)
Return a string representing the current date in the date format indicated by dateFormat. The dateFormat parameter can be 0=short, 1=long, 2=abbreviated.
¬†
Boolean **GetDirectoryDialog**(String *dirName*)
Puts up the GetDirectory dialog and stores the path of the chosen directory in 'dirName'
¬†
RGBImage **GetDisplayAsImage**(ImageReference)
Convert the display of image to an RGB image.
¬†
DocumentWindow **GetDocumentWindow**(Number *index*)
Gets the 'index'th document window.
¬†
DocumentWindow **GetDocumentWindowByTitle**(String *name*)
Gets the document window named 'name'.
¬†
void **GetEstimatedMinMax**(ImageReference, NumberVariable *minPtr*, NumberVariable *maxPtr*)
Store the current estimated minimum and maximum of the image into the minPtr and maxPtr variables.
¬†
String **GetExceptionDescription**( )
Return the message that would be displayed in the error dialog box for an exception as a string.
¬†
String **GetExceptionString**( )
Return the message that would be displayed in the error dialog box for an exception as a string.
¬†
TagGroup **GetFilesInDirectory**(String *path*, Number *search_flags*)
Returns a tag group containing a list of the file names in the directory 'dir_path'
¬†
Number **GetFileSize**(Number *file*)
Returns the size of the file.
¬†
DocumentWindow **GetFloatingWindow**(Number *index*)
Gets the 'index'th floating window.
¬†
Boolean **GetFourImages**(String *title*, ImageVariable *image1*, ImageVariable *image2*, ImageVariable *image3*, ImageVariable *image4*)
Puts up a dialog and allows the user to choose four images. Returns 1 for Ok and 0 for Cancel.
¬†
Boolean **GetFourImagesWithPrompt**(String *prompt*, String *title*, ImageVariable *image1*, ImageVariable *image2*, ImageVariable *image3*, ImageVariable *image4*)
Puts up a dialog with the given prompt and allows the user to choose four images. Returns 1 for Ok and 0 for Cancel.
¬†
Boolean **GetFourLabeledImagesWithPrompt**(String *prompt*, String *title*, String *label1*, ImageVariable *image1*, String *label2*, ImageVariable *image2*, String *label3*, ImageVariable *image3*, String *label4*,

ImageVariable *image4*)
Puts up a dialog with the given prompt and allows the user to choose four images. Returns 1 for Ok and 0 for Cancel.
¬†

Boolean **GetFrontImage**(ImageVariable)
Set the image variable to represent the foremost image, returns 1 if successful; return 0 otherwise.
¬†

ImageReference **GetFrontImage**( )
Return the foremost image.
¬†

ImageDocument **GetFrontImageDocument**( )
Returns the front image document.
¬†

Number **GetFrontImageID**( )
Return the id of the front most image window.
¬†

Number **GetImageDataSeed**(ImageReference)
Return a seed representing the data of the image. Each time the image data changes, the seed will change.
¬†

ImageDocument **GetImageDocument**(Number *position*)
Returns the image document by position with the application.
¬†

ImageDocument **GetImageDocumentByID**(Number *id*)
Returns the image document whose id is 'id'.
¬†

ImageReference **GetImageFromID**(Number *imageID*)
Return the image corresponding the imageID.
¬†

Boolean **GetImageFromID**(ImageVariable, Number *imageID*)
Store the image corresponding the imageID into the image variable. Return 1 if one is found; return 0 otherwise.
¬†

Number **GetImageID**(ImageReference)
Return the id of the image.
¬†

Boolean **GetInteger**(String *prompt*, Number *initalValue*, NumberVariable *result*)
Puts up a dialog with the given prompt and allows the user to enter an integer. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.
¬†

Boolean **GetInversionMode**(ImageReference)
Return the contrast inversion mode of the image (1=inverted, 0=not inverted).
¬†

Number **GetKey**( )
Returns the key that was last pressed.
¬†

String **GetLabel**(ImageReference)
Return the image label of the image.
¬†

void **GetLimits**(ImageReference, NumberVariable *lowPtr*, NumberVariable *highPtr*)
Stores display limits into the lowPtr and highPtr variables.
¬†

void **GetMaximalDocumentWindowRect**(Number *options*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the bounds of the content region of the largest document window.
¬†

String **GetName**(ImageReference)
Return the name of the image's image document.
¬†

ImageReference **GetNamedImage**(String *name*)
Return the image with the image document name.
¬†

Boolean **GetNamedImage**(ImageVariable, String *name*)
Store the image with the image document name into the image variable. Return 1 if one is found; return 0 otherwise.
¬†

Number **GetNextImageID**(Number *id*)
Return the id of the image window following the image with the given id.
¬†

Boolean **GetNoteState**(ImageReference, String *noteLabel*, NumberVariable *hidden*)
Stores whether the image note with the label noteLabel is in the copy-to-image list into the copyToImage parameter. The hidden parameter is no longer used.
¬†

Number **GetNthAnnotationID**(ImageReference, Number *index*)
Return the ID of the index'th annotation in the image.
¬†

DocumentWindow **GetNthDocumentWindowOfType**(Number *type*, Number *index*)
Returns the 'index'th document window of type 'type'.
¬†

Number **GetNthImageID**(Number *n*)
Return the id of the nth image (number from 0). The images are in no particular order.
¬†

Boolean **GetNthPersistentNoteTitle**(String *tagPath*, Number *index*, StringVariable *noteValue*)
Stores the title of the tag with the given index from the persistent tag group indicated by tagPath into noteValue.
¬†

String **GetNthPersistentNoteTitle**(String *tagPath*, Number *index*)
Returns the title of the tag with the given index from the persistent tag group indicated by tagPath.
¬†

void **GetNthPersistentNumberNote**(String *tagPath*, Number *index*, NumberVariable *noteValue*)
Stores the number with the given index from the persistent tag list indicated by tagPath into noteValue.
¬†

Boolean **GetNumber**(String *prompt*, Number *initalValue*, NumberVariable *result*)
Puts up a dialog with the given prompt and allows the user to enter a number. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.
¬†

Boolean **GetNumberNote**(ImageReference, String *noteLabel*, NumberVariable *noteValue*)
Stores the value of the image note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetOneImage**(String *title*, ImageVariable *image1*)
Puts up a dialog and allows the user to choose an image. Returns 1 for Ok and 0 for Cancel.
¬†

Boolean **GetOneImageWithPrompt**(String *prompt*, String *title*, ImageVariable *image1*)
Puts up a dialog and allows the user to choose an image. Returns 1 for Ok and 0 for Cancel.
¬†

Boolean **GetOneLabeledImageWithPrompt**(String *prompt*, String *title*, String *label1*, ImageVariable *image1*)
Puts up a dialog and allows the user to choose an image. Returns 1 for Ok and 0 for Cancel.
¬†

void **GetOrigin**(ImageReference, NumberVariable *x*, NumberVariable *y*)
Store the origin of image into the x and y variables. The origin is in the same units as scale.
¬†

Number **GetOSTickCount**( )
Return a tick count appropriate for the operating system.
¬†

Number **GetOSTicksPerSecond**( )
Return the number of ticks per second of a tick count appropriate for the operating system.
¬†

TagGroup **GetPackageTags**(String *identifier*)
Return the tags specified by identifier. The identifier is used to identify tags loaded with a specific package.
¬†

Boolean **GetPersistentComplexNumberNote**(String *noteLabel*, ComplexNumberVariable *noteValue*)
Stores the value of the persistent note with label noteLabel into the noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentLongNote**(String *noteLabel*, NumberVariable *noteValue*)
Stores the value of the persistent note with label noteLabel into the noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentNoteState**(String *noteLabel*, NumberVariable *copyToImage*, NumberVariable *hidden*)
Stores whether the note with the label noteLabel is in the copy-to-image list into the copyToImage parameter. The hidden parameter is no longer used.
¬†

Boolean **GetPersistentNumberNote**(String *noteLabel*, NumberVariable *noteValue*)
Stores the value of the persistent note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentPointNote**(String *noteLabel*, NumberVariable *x*, NumberVariable *y*)
Gets the value of the persistent note with label noteLabel into the x and y variables. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentRectNote**(String *noteLabel*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the value of the persistent note with label noteLabel into the top, left, bottom, and right variables. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentRGBNumberNote**(String *noteLabel*, RGBNumberVariable *noteValue*)
Stores the value of the persistent note with label noteLabel into the noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

Boolean **GetPersistentStringNote**(String *noteLabel*, String *noteBuffer*)
Copies the value of the persistent note with the label noteLabel into the noteBuffer variable. Returns 1 if the note exists returns 0 otherwise.
¬†

String **GetPersistentStringNote**(String *noteLabel*)
Returns the value of the persistent note with the label noteLabel. Returns an empty string if the note does not exist.
¬†

TagGroup **GetPersistentTagGroup**( )
Gets the persistent tag group.
¬†

RGBNumber **GetPixel**(RGBImage, Number *x*, Number *y*)
Return the pixel in the image at [x,y].
¬†

ComplexNumber **GetPixel**(ComplexImage, Number *x*, Number *y*)
Return the pixel in the image at [x,y]. Does not work on packed complex images.
¬†

Number **GetPixel**(RealImage, Number *x*, Number *y*)
Return the pixel in the image at [x,y].
¬†

Number **GetPlatformInfo**(Number *info*)
Return platform info. info=1 is general platform (1=MacOS,2=Windows).
¬†

Boolean **GetPointNote**(ImageReference, String *noteLabel*, NumberVariable *x*, NumberVariable *y*)
Gets the value of the image note with label noteLabel into the x and y variables. Returns 1 if the note exists returns 0 otherwise.
¬†

Number **GetProcessID**( )
Returns the current process id.
¬†

void **GetRawStreamPos**(Number *rawStream*, NumberVariable *pos*)
Store the current position in rawStream into the pos variable.
¬†

void **GetRawStreamSize**(Number *rawStream*, NumberVariable *size*)
Store the length of rawStream into the size variable.
¬†

Boolean **GetRectNote**(ImageReference, String *noteLabel*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the value of the image note with label noteLabel into the top, left, bottom, and right variables. Returns 1 if the note exists returns 0 otherwise.
¬†

DocumentWindow **GetResultsWindow**(Boolean *open*)
Gets the results window. If the window is not open, and 'open' is true, the results window is opened.
¬†

Boolean **GetRGBColorDialog**(String *prompt*, RGBNumber *defaultColor*, RGBNumberVariable *result*)
Puts up the Color Picker dialog with the given prompt and allows the user to choose a color. The default color is passed in and the resulting color is stored in the result parameter. Returns 1 for OK and 0 for Cancel.
¬†

Boolean **GetRGBNumberNote**(ImageReference, String *noteLabel*, RGBNumberVariable *noteValue*)
Stores the value of the image note with label noteLabel into noteValue variable. Returns 1 if the note exists returns 0 otherwise.
¬†

ROI **GetROIFromID**(Number *id*)
Returns the region of interest associated with the ID or NULL if it does not exist.
¬†

void **GetScale**(ImageReference, NumberVariable *x*, NumberVariable *y*)
Store the scale of image into the x and y variables.

¬†

`void` **`GetScreenSize`**`( NumberVariable` *`width`*`, NumberVariable` *`height`* `)`

Store the size of the screen into the width and height variables.

¬†

`ScriptObject` **`GetScriptObjectFromID`**`( Number` *`id`* `)`

Returns the script object associated with the ID or NULL if the object does not exist.

¬†

`Boolean` **`GetSelection`**`( ImageReference, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`

Stores the coordinates (in pixels) of the image's selection into the top, left, bottom, and right variables. Returns 1 if there was a selection and 0 if there wasn't.

¬†

`void` **`GetSize`**`( ImageReference, NumberVariable` *`width`*`, NumberVariable` *`height`* `)`

Store the width and height of the 2D image into the width and height variables.

¬†

`String` **`GetSpecialDirectory`**`( Number` *`index`* `)`

Return one of the special directories. 0=current directory, 1=executable directory.

¬†

`Number` **`GetSpecialWindow`**`( Number` *`index`* `)`

Return one of the special windows. On Windows, 0=frame window, 1=top-most dialog.

¬†

`Boolean` **`GetString`**`( String` *`prompt`*`, String` *`initalValue`*`, String` *`result`* `)`

Puts up a dialog with the given prompt and allows the user to enter a string. The initial value is passed as a parameter and the result in stored in result. Returns 1 for OK and 0 for Cancel.

¬†

`Boolean` **`GetStringFromList`**`( ImageReference, String` *`tagPath`*`, Number` *`index`*`, String` *`noteBuffer`* `)`

Copies the string with the given index from the image tag list indicated by tagPath to noteBuffer.

¬†

`String` **`GetStringFromList`**`( ImageReference, String` *`tagPath`*`, Number` *`index`* `)`

Returns the string with the given index from the image tag list indicated by tagPath.

¬†

`String` **`GetStringFromPersistentList`**`( String` *`tagPath`*`, Number` *`index`* `)`

Returns the string with the given index from the persistent tag list indicated by tagPath.

¬†

`Boolean` **`GetStringFromPersistentList`**`( String` *`tagPath`*`, Number` *`index`*`, String` *`noteBuffer`* `)`

Copies the string with the given index from the persistent tag list indicated by tagPath to noteBuffer.

¬†

`String` **`GetStringNote`**`( ImageReference, String` *`noteLabel`* `)`

Returns the value of the image note with the label noteLabel. Returns an empty string if the note does not exist.

¬†

`Boolean` **`GetStringNote`**`( ImageReference, String` *`noteLabel`*`, String` *`noteValue`* `)`

Copies the value of the image note with the label noteLabel into the noteBuffer variable. Returns 1 if the note exists returns 0 otherwise.

¬†

`Boolean` **`GetThreeImages`**`( String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`*`, ImageVariable` *`image3`* `)`

Puts up a dialog and allows the user to choose three images. Returns 1 for Ok and 0 for Cancel.

¬†

`Boolean` **`GetThreeImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`*`, ImageVariable` *`image3`* `)`

Puts up a dialog with the given prompt and allows the user to choose three images. Returns 1 for Ok and 0 for Cancel.

¬†

`Boolean` **`GetThreeLabeledImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, String` *`label1`*`, ImageVariable` *`image1`*`, String` *`label2`*`, ImageVariable` *`image2`*`, String` *`label3`*`, ImageVariable` *`image3`* `)`

Puts up a dialog with the given prompt and allows the user to choose three images. Returns 1 for Ok and 0 for Cancel.

¬†

`Number` **`GetTicks`**`( )`

Return the MacOS system tick count.

¬†

`Number` **`GetTicksPerSecond`**`( )`

Return the number of ticks per second.

¬†

`String` **`GetTime`**`( Boolean` *`wantSeconds`* `)`

Return a string representing the current time with or without seconds as indicated by the wantSeconds parameter.

¬†

`Boolean` **`GetTwoImages`**`( String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`* `)`

Puts up a dialog and allows the user to choose two images. Returns 1 for Ok and 0 for Cancel.
¬†
`Boolean` **`GetTwoImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, ImageVariable` *`image1`*`, ImageVariable` *`image2`*` )`
Puts up a dialog with the given prompt and allows the user to choose two images. Returns 1 for Ok and 0 for Cancel.
¬†
`Boolean` **`GetTwoLabeledImagesWithPrompt`**`( String` *`prompt`*`, String` *`title`*`, String` *`label1`*`, ImageVariable` *`image1`*`, String` *`label2`*`, ImageVariable` *`image2`*` )`
Puts up a dialog with the given prompt and allows the user to choose two images. Returns 1 for Ok and 0 for Cancel.
¬†
`Number` **`GetUnitsH`**`( ImageReference, Number` *`x`*` )`
Return the horizontal pixels x in calibrated units.
¬†
`String` **`GetUnitString`**`( ImageReference )`
Returns the unit string of the image.
¬†
`Number` **`GetUnitsV`**`( ImageReference, Number` *`y`*` )`
Return the vertical pixels y in calibrated units.
¬†
`void` **`GetWindowPosition`**`( ImageReference, NumberVariable` *`xPos`*`, NumberVariable` *`yPos`*` )`
Store the image's image document window position into the xPos and yPos variables.
¬†
`void` **`GetWindowSize`**`( ImageReference, NumberVariable` *`width`*`, NumberVariable` *`height`*` )`
Store the image's image document window size into the width and height variables.
¬†
`Number` **`GetZoom`**`( ImageReference )`
Return the zoom of the image display.
¬†
`void` **`GrabSemaphore`**`( Number )`
Grab the semaphore. Block until it is available. Used only with background processing.
¬†
`RealNumberExpression` **`green`**`( RGBNumberExpression )`
Return the green portion of an RGB number.
¬†
`void` **`GroupAnnotationUngroup`**`( Component` *`comp`*` )`
Ungroups the group annotation.
¬†
`Boolean` **`HasAcquisitionDaemon`**`( ImageReference )`
Returns 1 if the image has an attached daemon and returns 0 otherwise.
¬†
`String` **`Hex`**`( Number` *`n`*` )`
Returns the number as a hexidecimal string.
¬†
`String` **`Hex`**`( Number` *`n`*`, Number` *`length`*` )`
Returns the number as a hexidecimal string of the given length.
¬†
`void` **`HideImage`**`( ImageReference )`
Hide the image's image document.
¬†
`RealImageExpression` **`icol`**`( )`
¬†
`RealImageExpression` **`icolumn`**`( )`
¬†
`RealImageExpression` **`idepth`**`( )`
¬†
`void` **`if`**`( RealNumber, Expression, Expression )`
¬†
`void` **`if`**`( RealNumber, Expression )`
¬†
`ComplexImage` **`IFFT`**`( ComplexImage` *`source`*` )`
Creates a new complex 8-byte image from the inverse FFT of the complex image source.
¬†
`RealImageExpression` **`iheight`**`( )`
¬†
`void` **`ImageCalculateHistogram`**`( ImageReference` *`image`*`, ImageReference` *`hist_image`*`, Number` *`complexMode`*`,`

Number *min*, Number *max*)
Calculates the histogram of 'image', mapping [min,max] into 'hist_image'.
¬†
void **ImageCalculateMinMax**(ImageReference *image*, Number *surveyTechnique*, Number *complexMode*, NumberVariable *min*, NumberVariable *max*)
Calculates the minimum and maximum value of 'image' using survey technique 'surveyTechnique'.
¬†
ImageReference **ImageClone**(ImageReference *img*)
Returns a clone of 'img'.
¬†
void **ImageCopyCalibrationFrom**(ImageReference *image*, ImageReference *src_image*)
Copy the calibration of 'src_image' to 'image'.
¬†
Number **ImageCountImageDisplays**(ImageReference)
Returns the number of image displays in which this image is displayed.
¬†
Number **ImageCountImageDisplaysInImageDocument**(ImageReference, ImageDocument *imgDoc*)
Returns the number of image displays in the image document in which this image is displayed.
¬†
ImageDisplay **ImageCreateImageDisplay**(ImageReference, Number *displayType*)
Creates a new image display of type 'displayType' for the image.
¬†
void **ImageData_CacheChanged**(Number *imageDataID*)
Tell the array processor manager that the cached image has changed. See the SDK documentation for more information.
¬†
void **ImageData_Changed**(Number *imageDataID*)
Tell the array processor manager that the image has changed. See the SDK documentation for more information.
¬†
void **ImageData_FlushCache**(Number *imageDataID*)
Tell the array processor manager to flush the image from the cache. See the SDK documentation for more information.
¬†
Number **ImageData_GetSeed**(Number *imageDataID*)
Ask the array processor manager for the data seed of the image. See the SDK documentation for more information.
¬†
Boolean **ImageData_IsLocalCopyValid**(Number *imageDataID*)
Ask the array processor manager if the local copy is valid or not. See the SDK documentation for more information.
¬†
void **ImageData_SetLocalSeed**(Number *imageDataID*, Number *value*)
Tell the array processor manager to set the local data seed. See the SDK documentation for more information.
¬†
void **ImageDisplayAccumulateROIsToMask**(ImageDisplay *imgDisp*, ImageReference *mask*, Number *top*, Number *left*, Number *bottom*, Number *right*, Number *mask_val*)
Sets mask to mask_val at points in imageDisplay's rois
¬†
void **ImageDisplayAddKeyListener**(ImageDisplay *imgDisp*, String *listener_key*, String *listener_script*)
Adds the listener_script to the key listener list under the tag listener_key.
¬†
void **ImageDisplayAddROI**(ImageDisplay *imgDisp*, ROI *roi*)
Adds the roi to this image display.
¬†
void **ImageDisplayChangeDisplayType**(ImageDisplay *imgDisp*, Number *new_type*)
Changes the type of the image display.
¬†
Number **ImageDisplayCountROIs**(ImageDisplay *imgDisp*)
Returns the number of ROIs on this image display.
¬†
void **ImageDisplayDeleteROI**(ImageDisplay *imgDisp*, ROI *roi*)
Deletes the roi from this image display.
¬†
Boolean **ImageDisplayDoesROIExist**(ImageDisplay *imgDisp*, String *name*)
Determines whether the given ROI exists on this image display.
¬†
void **ImageDisplayExportToFile**(ImageDisplay *imgDisp*, String *format*, String *file_name*)
Exports the display to the file 'file_name' using the format 'format'.

¬†
ImageReference **ImageDisplayGetBufferedImage**(ImageDisplay *imgDisp*)
Gets the image resulting from the contrast transformation.
¬†
Number **ImageDisplayGetComplexMode**(ImageDisplay *imgDisp*)
Gets the complex mode of the display.
¬†
Number **ImageDisplayGetComplexModeRange**(ImageDisplay *imgDisp*)
Gets the parameter used in converting complex to real.
¬†
void **ImageDisplayGetContrastLimits**(ImageDisplay *imgDisp*, NumberVariable *low*, NumberVariable *high*)
Gets the contrast limits of the display.
¬†
Number **ImageDisplayGetContrastMode**(ImageDisplay *imgDisp*)
Returns the contrast mode.
¬†
void **ImageDisplayGetContrastParameters**(ImageDisplay *imgDisp*, NumberVariable *bright*, NumberVariable *contrast*)
Gets the parameters for the contrast mode.
¬†
ImageReference **ImageDisplayGetDisplayedImage**(ImageDisplay *imgDisp*)
Gets the image that is actually displayed in the image display.
¬†
void **ImageDisplayGetDisplayedLayers**(ImageDisplay *imgDisp*, NumberVariable *start*, NumberVariable *end*)
Gets the layers that are summed into the display.
¬†
Number **ImageDisplayGetDisplayType**(ImageDisplay *imgDisp*)
Returns type of the image display.
¬†
Boolean **ImageDisplayGetDoAutoSurvey**(ImageDisplay *imgDisp*)
Determines whether min and max are determined automatically.
¬†
ImageReference **ImageDisplayGetExportImage**(ImageDisplay *imgDisp*, Number *mode*, ImageDisplay *clut_display*)
Gets the image representation of the image as it appears on the screen at full resolution.
¬†
ImageReference **ImageDisplayGetImage**(ImageDisplay *imgDisp*)
Returns the single image displayed by the image display.
¬†
void **ImageDisplayGetImageAdjustRect**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Returns the image display outside the image rect
¬†
void **ImageDisplayGetImageAdjustRectInView**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Returns the image display outside the image rect in view coordinates
¬†
void **ImageDisplayGetImageRect**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the bounds of the image in the image display.
¬†
void **ImageDisplayGetImageRectInView**(ImageDisplay *imgDisp*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the bounds of the image in the image display in view coordinates.
¬†
ImageReference **ImageDisplayGetInputColorTable**(ImageDisplay *imgDisp*)
Gets the input color table for the display.
¬†
ImageReference **ImageDisplayGetIntensityTransformation**(ImageDisplay *imgDisp*)
Gets the ITT of the display.
¬†
Number **ImageDisplayGetMinimumContrast**(ImageDisplay *imgDisp*)
Gets the minimum contrast of the display.
¬†
ImageReference **ImageDisplayGetOutputColorTable**(ImageDisplay *imgDisp*)

Gets the output color table for the display.
¬†
ROI **ImageDisplayGetROI**(ImageDisplay *imgDisp*, Number *index*)
Returns the given ROI on this image display.
¬†
Number **ImageDisplayGetROISelectionStyle**(ImageDisplay *imgDisp*, ROI *r*)
Gets the selection style of the roi in the imag display.
¬†
Number **ImageDisplayGetSurveyTechnique**(ImageDisplay *imgDisp*)
Gets the survey technique of the display.
¬†
Boolean **ImageDisplayIsCaptionOn**(ImageDisplay *imgDisp*)
Returns true if captions are drawn.
¬†
Boolean **ImageDisplayIsROISelected**(ImageDisplay *imgDisp*, ROI *roi*)
Determines whether the given ROI is selected on this image display.
¬†
Boolean **ImageDisplayIsValid**(ImageDisplay *imgDisp*)
Returns true if 'imageDisplay' points to a valid object.
¬†
ROI **ImageDisplayLookupROI**(ImageDisplay *imgDisp*, String *name*)
Returns the given ROI on this image display.
¬†
ROI **ImageDisplayLookupROIByID**(ImageDisplay *imgDisp*, Number *id*)
Returns the ROI with the given id on this image display.
¬†
ImageDisplay **ImageDisplayNullify**(!)
¬†
void **ImageDisplayRemoveKeyListener**(ImageDisplay *imgDisp*, String *listener_key*)
Removes the listener script with the tag listener_key from the key listener list.
¬†
void **ImageDisplaySetCaptionOn**(ImageDisplay *imgDisp*, Boolean *on*)
Sets whether to draw captions.
¬†
void **ImageDisplaySetComplexMode**(ImageDisplay *imgDisp*, Number *mode*)
Sets the complex mode of the display.
¬†
void **ImageDisplaySetComplexModeRange**(ImageDisplay *imgDisp*, Number *range*)
Sets the parameter used in converting complex to real.
¬†
void **ImageDisplaySetContrastLimits**(ImageDisplay *imgDisp*, Number *low*, Number *hight*)
Sets the contrast limits of the display.
¬†
void **ImageDisplaySetContrastMode**(ImageDisplay *imgDisp*, Number *mode*)
Sets the contrast mode.
¬†
void **ImageDisplaySetContrastParameters**(ImageDisplay *imgDisp*, Number *bright*, Number *contrast*)
Gets the parameters for the contrast mode.
¬†
void **ImageDisplaySetDisplayedLayers**(ImageDisplay *imgDisp*, Number *start*, Number *end*)
Sets the layers that are summed into the display.
¬†
void **ImageDisplaySetDoAutoSurvey**(ImageDisplay *imgDisp*, Boolean *do_survey*)
Sets whether min and max are determined automatically.
¬†
void **ImageDisplaySetImageRect**(ImageDisplay *imgDisp*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the bounds of the image part of the image display.
¬†
void **ImageDisplaySetInputColorTable**(ImageDisplay *imgDisp*, ImageReference *clut*)
Sets the input color table of the display.
¬†
void **ImageDisplaySetIntensityTransformation**(ImageDisplay *imgDisp*, ImageReference *itt*)
Sets the ITT of the display.
¬†

void **ImageDisplaySetMinimumContrast**(ImageDisplay *imgDisp*, Number *contrast*)
Sets the minimum contrast of the display.
¬†
void **ImageDisplaySetROISelected**(ImageDisplay *imgDisp*, ROI *roi*, Boolean *select*)
Sets the selection status of the region of interest in the image display.
¬†
void **ImageDisplaySetROISelectionStyle**(ImageDisplay *imgDisp*, ROI *r*, Number *style*)
Sets the selection style of the roi in the imag display.
¬†
void **ImageDisplaySetSurveyTechnique**(ImageDisplay *imgDisp*, Number *tech*)
Sets the survey technique of the display.
¬†
void **ImageDocumentAddImage**(ImageDocument *imgDoc*, ImageReference *image*)
Adds the given image to the list maintained in the image document.
¬†
ImageDisplay **ImageDocumentAddImageDisplay**(ImageDocument *imgDoc*, ImageReference *image*, Number *displayType*)
Adds the given image and an image display for it of the given type.
¬†
void **ImageDocumentAddToUserInterface**(ImageDocument *imgDoc*)
Places the image document in the list of user interface documents.
¬†
void **ImageDocumentClean**(ImageDocument *imgDoc*)
Marks the image document as clean (doesn't need to be saved).
¬†
ImageDocument **ImageDocumentClone**(ImageDocument *imgDoc*, Boolean *doDeepCopy*)
Returns a duplicate of the image docuemnt, creating a copy of its images if 'doDeepCopy' is true.
¬†
void **ImageDocumentClose**(ImageDocument *imgDoc*, Boolean *saving*)
Closes the given image document. If saving is true then asks whether to save it, otherwise just closes it.
¬†
Number **ImageDocumentCountImages**(ImageDocument *imgDoc*)
Returns the number of images contained in this image document.
¬†
ImageReference **ImageDocumentCreateRGBImageFromDocument**(ImageDocument *imgDoc*, Number *width*, Number *height*, Number *extract_style*, Number *constraints*)
Creates an image by scaling the image document into (width, height).
¬†
void **ImageDocumentDeleteImage**(ImageDocument *imgDoc*, ImageReference *image*)
Deletes the given image from this image document.
¬†
Boolean **ImageDocumentDoesImageWithIDExist**(ImageDocument *imgDoc*, Number *id*)
Determines whether the image with the given id exists within this image document.
¬†
void **ImageDocumentEnsurePlacedOnPage**(ImageDocument *imgDoc*)
Makes sure the document has been layed out on the physical page.
¬†
Number **ImageDocumentGetAsPICT**(ImageDocument *imgDoc*)
Returns this image as a PICT.
¬†
Component **ImageDocumentGetComponentByID**(ImageDocument *imgDoc*, Number *id*)
Returns an annotation contained in this image document by id.
¬†
Number **ImageDocumentGetID**(ImageDocument *imgDoc*)
Gets the id of the image document.
¬†
ImageReference **ImageDocumentGetImage**(ImageDocument *imgDoc*, Number *position*)
Returns the image contained within this image document by position.
¬†
ImageReference **ImageDocumentGetImageByID**(ImageDocument *imgDoc*, Number *id*)
Returns an image contained in this image document by id.
¬†
ImageDisplay **ImageDocumentGetImageModeDisplay**(ImageDocument *imgDoc*)
Gets the image display targeted by the current image mode.
¬†

void **ImageDocumentGetMinimumPointSize**(ImageDocument *imgDoc*, NumberVariable *x*, NumberVariable *y*)
Gets the size of the minimum point in view coordinates.
¬†
String **ImageDocumentGetName**(ImageDocument *imgDoc*)
Returns the name of the image document.
¬†
void **ImageDocumentGetPageBounds**(ImageDocument *imgDoc*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the page bounds of the document in page coordinates.
¬†
void **ImageDocumentGetPageResolution_72dpi**(ImageDocument *imgDoc*, NumberVariable *horz*, NumberVariable *vert*)
Returns the resolution of page coordinates in 72 dots per inch (returns page units per dot).
¬†
void **ImageDocumentGetPageResolution_Printer**(ImageDocument *imgDoc*, NumberVariable *horz*, NumberVariable *vert*)
Returns the resolution of page coordinates in printer pixels (returns page units per printer pixel).
¬†
void **ImageDocumentGetPaperBounds**(ImageDocument *imgDoc*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the paper bounds of the document in page coordinates.
¬†
void **ImageDocumentGetPreferredViewRect**(ImageDocument *imgDoc*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets rectangle in view coordinates of the area that is by default displayed in this mode.
¬†
void **ImageDocumentGetReferencePointSize**(ImageDocument *imgDoc*, NumberVariable *x*, NumberVariable *y*)
Gets the size of the reference point in view coordinates.
¬†
Component **ImageDocumentGetRootComponent**(ImageDocument *imgDoc*)
Gets the root annotation of the image document.
¬†
TagGroup **ImageDocumentGetTagGroup**(ImageDocument *imgDoc*)
Gets the tag group associated with the image document.
¬†
void **ImageDocumentGetUnzoomedPointSize**(ImageDocument *imgDoc*, NumberVariable *x*, NumberVariable *y*)
Gets the size of the unzoomed point in view coordinates.
¬†
void **ImageDocumentGetViewExtent**(ImageDocument *imgDoc*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the extent in view coordinates of the items visible in the current view mode.
¬†
void **ImageDocumentGetViewToWindowTransform**(ImageDocument *imgDoc*, NumberVariable *off_x*, NumberVariable *off_y*, NumberVariable *scale_x*, NumberVariable *scale_y*)
Returns the transformation from view to screen coordinates.
¬†
void **ImageDocumentGetVisibleViewRect**(ImageDocument *imgDoc*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the view coordinates of the rectangle visible in the view.
¬†
DocumentWindow **ImageDocumentGetWindow**(ImageDocument *imgDoc*)
Returns the window displaying the document.
¬†
Boolean **ImageDocumentHasBeenPlacedOnPage**(ImageDocument *imgDoc*)
Returns 'true' if the document has been layed out within the physical page.
¬†
void **ImageDocumentHide**(ImageDocument *imgDoc*)
Hides the given image document.
¬†
Boolean **ImageDocumentIsInImageMode**(ImageDocument *imgDoc*)
Returns true if the view of the document is in image mode.
¬†
Boolean **ImageDocumentIsInPageMode**(ImageDocument *imgDoc*)
Returns true if the view of the document is in page mode.
¬†
Boolean **ImageDocumentIsValid**(ImageDocument *imgDoc*)

Returns true if 'imageDocument' points to a valid object.
¬†
`void` **`ImageDocumentMaximizeRectInView`**`( ImageDocument` *`imgDoc`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Zooms the view so the rectangle is centered and maximal.
¬†
`ImageDocument` **`ImageDocumentNullify`**`( ! )`
¬†
`Boolean` **`ImageDocumentPrint`**`( ImageDocument` *`imgDoc`*` )`
Print the image document, returning 'true' if successful.
¬†
`void` **`ImageDocumentRemoveFromUserInterface`**`( ImageDocument` *`imgDoc`*` )`
Removes the image document from the list of user interface documents.
¬†
`void` **`ImageDocumentSaveToFile`**`( ImageDocument` *`imgDoc`*`, String` *`handler`*`, String` *`fileName`*` )`
Saves the image document to the given file name using the I/O handler specified.
¬†
`void` **`ImageDocumentSetCurrentViewAsUnzoomed`**`( ImageDocument` *`imgDoc`*` )`
Makes the current view the unzoomed view.
¬†
`void` **`ImageDocumentSetName`**`( ImageDocument` *`imgDoc`*`, String` *`name`*` )`
Sets the name of the image document.
¬†
`void` **`ImageDocumentSetRectInView`**`( ImageDocument` *`imgDoc`*`, Number` *`v_t`*`, Number` *`v_l`*`, Number` *`v_b`*`, Number` *`v_r`*`, Number` *`w_t`*`, Number` *`w_l`*`, Number` *`w_b`*`, Number` *`w_r`*` )`
Zooms the view so the view rect (v_l,v_t,v_b,v_r) is displayed in the window rect (w_l,w_t,w_b,w_r).
¬†
`DocumentWindow` **`ImageDocumentShow`**`( ImageDocument` *`imgDoc`*` )`
Shows the given image document.
¬†
`DocumentWindow` **`ImageDocumentShowAtPosition`**`( ImageDocument` *`imgDoc`*`, Number` *`x`*`, Number` *`y`*` )`
Shows the given image document at the application position (x,y).
¬†
`DocumentWindow` **`ImageDocumentShowAtRect`**`( ImageDocument` *`imgDoc`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Shows the given image document at the rect (top,left,bottom,right).
¬†
`void` **`ImageDocumentSwitchToImageMode`**`( ImageDocument` *`imgDoc`*`, ImageDisplay` *`imgDisp`*` )`
Switches the view of the document to image mode focused on the display 'imgDisp'.
¬†
`void` **`ImageDocumentSwitchToPageMode`**`( ImageDocument` *`imgDoc`*` )`
Switches the view of the document to page mode.
¬†
`void` **`ImageDocumentUpdateDisplay`**`( ImageDocument` *`imgDoc`*` )`
Updates the display of the image document.
¬†
`Number` **`ImageGetDataElementBitSize`**`( ImageReference` *`img`*` )`
Returns the size of the data elements in bits.
¬†
`Number` **`ImageGetDataElementByteSize`**`( ImageReference` *`img`*` )`
Returns the smallest number of bytes that can hold a data element.
¬†
`Number` **`ImageGetDataSeed`**`( ImageReference` *`img`*` )`
Gets the seed of the image data.
¬†
`Number` **`ImageGetDataType`**`( ImageReference` *`img`*` )`
Returns a long representing the data type.
¬†
`String` **`ImageGetDescriptionText`**`( ImageReference` *`img`*` )`
Gets the description text associated with the image.
¬†
`void` **`ImageGetDimensionCalibration`**`( ImageReference, Number` *`dimension`*`, NumberVariable` *`origin`*`, NumberVariable` *`scale`*`, String` *`units`*`, Number` *`calibrationFormat`*` )`
Gets the calibration information of the given dimension.
¬†

Number **ImageGetDimensionOrigin**(ImageReference, Number *dimension*)
Returns the origin of the given dimension of image.
¬†
Number **ImageGetDimensionScale**(ImageReference, Number *dimension*)
Returns the scale of the given dimension of image.
¬†
Number **ImageGetDimensionSize**(ImageReference, Number *dimension*)
Gets the size of the given dimension.
¬†
void **ImageGetDimensionUnitInfo**(ImageReference, Number *dimension*, String *canon_units*, NumberVariable *power*)
Copies the unit string of the given dimension of image to the buffer.
¬†
String **ImageGetDimensionUnitString**(ImageReference, Number *dimension*)
Copies the unit string of the given dimension of image to the buffer.
¬†
Number **ImageGetID**(ImageReference)
Returns a unique identifier for the image.
¬†
ImageDisplay **ImageGetImageDisplay**(ImageReference, Number *index*)
Returns the given image display in which this image is displayed.
¬†
ImageDisplay **ImageGetImageDisplayInImageDocument**(ImageReference, ImageDocument *imgDoc*, Number *index*)
Returns the given image display in the image document in which this image is displayed.
¬†
Number **ImageGetIntensityOrigin**(ImageReference)
Returns the origin of image's intensity.
¬†
Number **ImageGetIntensityScale**(ImageReference)
Returns the scale of image's intensity.
¬†
void **ImageGetIntensityUnitInfo**(ImageReference, String *canon_units*, NumberVariable *power*)
Copies the unit string of image's intensity to the buffer.
¬†
String **ImageGetIntensityUnitString**(ImageReference)
Returns the units of the image's intensity.
¬†
String **ImageGetLabel**(ImageReference *img*)
Gets the label of the image as used in scripts.
¬†
String **ImageGetName**(ImageReference *img*)
Gets the name of the image.
¬†
Number **ImageGetNumDimensions**(ImageReference)
Returns number of dimensions of the image.
¬†
ImageDocument **ImageGetOrCreateImageDocument**(ImageReference *im*)
Returns an image document containing the image, creating one if necessary.
¬†
TagGroup **ImageGetTagGroup**(ImageReference *img*)
Gets the tags associated with the image.
¬†
ScriptObject **ImageGetUniqueID**(ImageReference *image*)
Returns the unique ID for this image. This id is globally unique across sessions and locations.
¬†
Boolean **ImageIsDataTypeBinary**(ImageReference *img*)
Returns true if the data in the image is binary.
¬†
Boolean **ImageIsDataTypeComplex**(ImageReference *img*)
Returns true if the data in the image is complex.
¬†
Boolean **ImageIsDataTypeFloat**(ImageReference *img*)
Returns true if the data in the image is floating point.
¬†

Boolean **ImageIsDataTypeInteger**(ImageReference *img*)
Returns true if the data in the image is integral.
¬†
Boolean **ImageIsDataTypePackedComplex**(ImageReference *img*)
Returns true if the data in the image is complex.
¬†
Boolean **ImageIsDataTypeReal**(ImageReference *img*)
Returns true if the data in the image is real.
¬†
Boolean **ImageIsDataTypeRGB**(ImageReference *img*)
Returns true if the data in the image is rgb.
¬†
Boolean **ImageIsDataTypeSignedInteger**(ImageReference *img*)
Returns true if the data in the image is integral and signed.
¬†
Boolean **ImageIsDataTypeUnsignedInteger**(ImageReference *img*)
Returns true if the data in the image is integral and unsigned.
¬†
Boolean **ImageIsDimensionCalibrationDisplayed**(ImageReference *im*, Number *dim*)
Returns 'true' if the calibration of the 'dim'th dimension is displayed.
¬†
Boolean **ImageIsIntensityCalibrationDisplayed**(ImageReference *im*)
Returns 'true' if the calibration of the intensity is displayed.
¬†
Boolean **ImageIsValid**(ImageReference *image*)
Returns true if 'image' is a valid object.
¬†
ImageReference **ImageNullify**(!)
¬†
void **ImageReadImageDataFromStream**(ImageReference *image*, ScriptObject *stream*, Number *stream_endianness*)
Reads image data from the stream. The image data type determines how the data is read, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
void **ImageSetDescriptionText**(ImageReference *img*, String *description*)
Sets the description text associated with the image.
¬†
void **ImageSetDimensionCalibration**(ImageReference, Number *dimenstion*, Number *origin*, Number *scale*, String *unitString*, Number *calibrationFormat*)
Sets the calibration for the given dimension.
¬†
void **ImageSetDimensionCalibrationDisplayed**(ImageReference *im*, Number *dim*, Boolean *do_display*)
Sets whether or not to display the 'dim'th dimension in calibrated units to 'do_display'.
¬†
void **ImageSetDimensionOrigin**(ImageReference, Number *dimension*, Number *origin*)
Sets the origin of the given dimension of image.
¬†
void **ImageSetDimensionScale**(ImageReference, Number *dimension*, Number *scale*)
Sets the scale of the given dimension of image.
¬†
void **ImageSetDimensionUnitInfo**(ImageReference, Number *dimension*, String *canon_units*, Number *power*)
Sets the unit string of the given dimension of image.
¬†
void **ImageSetDimensionUnitString**(ImageReference, Number *dimension*, String *units*)
Sets the unit string of the given dimension of image.
¬†
void **ImageSetIntensityCalibrationDisplayed**(ImageReference *im*, Boolean *do_display*)
Sets whether or not to display the intensity in calibrated units to 'do_display'.
¬†
void **ImageSetIntensityOrigin**(ImageReference, Number *origin*)
Sets the origin of image's intensity.
¬†
void **ImageSetIntensityScale**(ImageReference, Number *scale*)
Sets the scale of image's intensity.

¬†

`void` **`ImageSetIntensityUnitInfo`**`( ImageReference, String canon_units, Number power )`

Sets the unit string of image's intensity.

¬†

`void` **`ImageSetIntensityUnitString`**`( ImageReference, String units )`

Sets the unit string of image's intensity.

¬†

`void` **`ImageSetName`**`( ImageReference img, String name )`

Sets the name of the image.

¬†

`void` **`ImageSwapImageDataByteOrder`**`( ImageReference )`

Swaps the byte order for each long word in the image. ABCD become DCBA.

¬†

`ImageDocument` **`ImageWindowGetImageDocument`**`( DocumentWindow window )`

Gets the image document displayed in the window.

¬†

`void` **`ImageWriteImageDataToStream`**`( ImageReference image, ScriptObject stream, Number stream_endianness )`

Writes image data to the stream. The image data type determines how the data is written, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.

¬†

`RealNumberExpression` **`imaginary`**`( ComplexNumberExpression )`

Return the imaginary portion of the complex number.

¬†

`RealImageExpression` **`Index`**`( RealImage, RealImageExpression x, RealImageExpression y )`

Returns a pixel in the given real image at the position [x,y]. Performs bounds checking.

¬†

`RealImageExpression` **`Index`**`( RealImage, RealImageExpression x, RealImageExpression y, RealImageExpression z )`

Returns a pixel in the given real image at the position [x,y,z]. Performs bounds checking.

¬†

`ComplexImageExpression` **`Index`**`( ComplexImage, RealImageExpression x, RealImageExpression y, RealImageExpression z )`

Returns a pixel in the given Complex image at the position [x,y,z]. Performs bounds checking.

¬†

`RGBImageExpression` **`Index`**`( RGBImage, RealImageExpression x, RealImageExpression y, RealImageExpression z )`

Returns a pixel in the given RGB image at the position [x,y,z]. Performs bounds checking.

¬†

`ComplexImageExpression` **`Index`**`( ComplexImage, RealImageExpression x, RealImageExpression y )`

Returns a pixel in the given complex image at the position [x,y]. Performs bounds checking.

¬†

`RGBImageExpression` **`Index`**`( RGBImage, RealImageExpression x, RealImageExpression y )`

Returns a pixel in the given RGB image at the position [x,y]. Performs bounds checking.

¬†

`Number` **`InstallScriptLibraryFile`**`( String fileName )`

Loads the script file indicated by fileName, executes it, and publishes any functions contained inside. Always returns 0.

¬†

`RealImage` **`IntegerImage`**`( String title, Number bytes, Boolean isSigned, Number d0 )`

Creates a 1D integer image of size [d0] with the given title. The bytes and isSigned parameters specify integer specific attributes of the data.

¬†

`RealImage` **`IntegerImage`**`( String title, Number bytes, Boolean isSigned, Number d0, Number d1, Number d2 )`

Creates a 3D integer image of size [d0,d1,d2] with the given title. The bytes can be 1, 2, or 4 and isSigned can be 1 (true) or 0 (false).

¬†

`RealImage` **`IntegerImage`**`( String title, Number bytes, Boolean isSigned, Number d0, Number d1 )`

Creates a 2D integer image of size [d0,d1] with the given title. The bytes and isSigned parameters specify integer specific attributes of the data.

¬†

`Number` **`integrate`**`( RealImageExpression )`

Return the sum of all pixel values of the image expression.

¬†

`RealImageExpression` **`iplane`**`( )`

¬†

RealImageExpression **ipoints**( )
¬†
RealImageExpression **iradius**( )
¬†
RealImageExpression **irow**( )
¬†
Boolean **IsAnnotationSelected**(ImageReference, Number *annotationID*)
Return 1 if the annotation indicated by the annotationID within the image is selected; returns 0 otherwise.
¬†
Boolean **IsBinaryDataType**(ImageReference)
Returns 1 if the image is an binary data type; returns 0 otherwise.
¬†
Boolean **IsByteImage**(ImageReference)
Returns 1 if the image is unsigned 1-byte integer data; returns 0 otherwise.
¬†
Boolean **IsComplexDataType**(ImageReference, Number *bytes*)
Returns 1 if the image is an complex data type of size bytes; returns 0 otherwise.
¬†
Boolean **IsComplexImage**(ImageReference)
Returns 1 if the image is single precision complex data; returns 0 otherwise.
¬†
Boolean **IsDisplayValid**(ImageReference)
Return 1 if the image's display is up-to-date and 0 otherwise.
¬†
Boolean **IsExceptionUserAbort**( )
Returns true if the exception currently in effect is a user abort.
¬†
Boolean **IsFloatImage**(ImageReference)
Returns 1 if the image is single precision real data; returns 0 otherwise.
¬†
Boolean **IsImageComplex**(Number *id*)
Return true or false to indicate whether given image with the given id is complex-valued or not.
¬†
Boolean **IsImageReal**(Number *id*)
Return true or false to indicate whether given image with the given id is real-valued or not.
¬†
Boolean **IsImageRGB**(Number *id*)
Return true or false to indicate whether given image with the given id is RGB-valued or not.
¬†
Boolean **IsInfinite**(Number *v*)
Returns true if the value is infinite.
¬†
Boolean **IsIntegerDataType**(ImageReference, Number *bytes*, Boolean *isSigned*)
Returns 1 if the image is an integer data type of size bytes with a matching sign charateristic as signed; returns 0 otherwise.
¬†
Boolean **IsLongImage**(ImageReference)
Returns 1 if the image is signed 4-byte integer data; returns 0 otherwise.
¬†
Boolean **IsNan**(Number *v*)
Returns true if the value is not a number.
¬†
Boolean **IsPackedComplexImage**(ImageReference)
Returns 1 if the image is packed complex data; returns 0 otherwise.
¬†
Boolean **IsRealDataType**(ImageReference, Number *bytes*)
Returns 1 if the image is an real data type of size bytes; returns 0 otherwise.
¬†
Boolean **IsRGBDataType**(ImageReference, Number *bytes*)
Returns 1 if the image is an RGB data type of size bytes; returns 0 otherwise.
¬†
Boolean **IsShortImage**(ImageReference)
Returns 1 if the image is signed 2-byte integer data; returns 0 otherwise.
¬†
RealImageExpression **itheta**( )
¬†

RealImageExpression **iwidth**( )
¬†
void **KeepImage**(ImageReference)
Keep the image from being deleted automatically when the image's script scope is exited.
¬†
void **KillProcess**(Number *pid*)
Kills the process specified by pid.
¬†
String **left**(String, Number *count*)
Returns leftmost count characters of a string.
¬†
RealNumberExpression **LegendrePolynomial**(Number, Number, Number)
Return the Legendre polynomial function for a number.
¬†
Number **len**(String)
Returns the length of a string.
¬†
Number **LinePlotImageDisplayCountSlices**(LinePlotImageDisplay *lpid*)
Returns the number of slices in the line plot.
¬†
Number **LinePlotImageDisplayGetBaseIntensity**(LinePlotImageDisplay *lpid*)
Returns the base intensity of the line plot.
¬†
void **LinePlotImageDisplayGetContrastLimits**(LinePlotImageDisplay *lpid*, NumberVariable *lowLimit*, NumberVariable *highLimit*)
Gets the lowest and higest intensities displayed.
¬†
void **LinePlotImageDisplayGetDisplayedChannels**(LinePlotImageDisplay *lpid*, NumberVariable *leftChannel*, NumberVariable *rightChannel*)
Gets the leftmost and rightmost displayed channels.
¬†
void **LinePlotImageDisplayGetDoAutoSurvey**(LinePlotImageDisplay *lpid*, NumberVariable *doAutoSurveyLow*, NumberVariable *doAutoSurveyHigh*)
Gets whether to auto-survey is done on the high and low intensity limits.
¬†
Number **LinePlotImageDisplayGetSlice**(LinePlotImageDisplay *lpid*)
Returns slice currently displayed at the bottom.
¬†
void **LinePlotImageDisplayGetSliceComponentColor**(LinePlotImageDisplay *lpid*, Number *slice_index*, Number *comp_index*, NumberVariable *r*, NumberVariable *g*, NumberVariable *b*)
Returns the color of the 'comp_index'th component of the 'slice_index'th slice.
¬†
Number **LinePlotImageDisplayGetSliceDrawingStyle**(LinePlotImageDisplay *lpid*, Number *slice_index*)
Returns the drawing style of the 'slice_index'th slice.
¬†
void **LinePlotImageDisplayGetTrackingStyle**(LinePlotImageDisplay *lpid*, NumberVariable *track_style_x*, NumberVariable *track_style_y*)
Gets the tracking style of the line plot.
¬†
Boolean **LinePlotImageDisplayIsBackgroundOn**(LinePlotImageDisplay *lpid*)
Returns true if the background is erased.
¬†
Boolean **LinePlotImageDisplayIsFilled**(LinePlotImageDisplay *lpid*)
Returns true if the line plot is filled.
¬†
Boolean **LinePlotImageDisplayIsFrameOn**(LinePlotImageDisplay *lpid*)
Returns true if the frame is drawn.
¬†
Boolean **LinePlotImageDisplayIsGridOn**(LinePlotImageDisplay *lpid*)
Returns true if the grid is displayed on.
¬†
LinePlotImageDisplay **LinePlotImageDisplayNullify**(!)
¬†
void **LinePlotImageDisplaySetBackgroundOn**(LinePlotImageDisplay *lpid*, Boolean *on*)
Sets whether to erase the background.
¬†

void **LinePlotImageDisplaySetBaseIntensity**(LinePlotImageDisplay *lpid*, Number *base_intensity*)
Sets the base intensity of the line plot.
¬†
void **LinePlotImageDisplaySetContrastLimits**(LinePlotImageDisplay *lpid*, Number *lowLimit*, Number *highLimit*)
Sets the lowest and highest intensities displayed.
¬†
void **LinePlotImageDisplaySetDisplayedChannels**(LinePlotImageDisplay *lpid*, Number *leftChannel*, Number *rightChannel*)
Sets the leftmost and rightmost displayed channels.
¬†
void **LinePlotImageDisplaySetDoAutoSurvey**(LinePlotImageDisplay *lpid*, Boolean *doAutoSurveyLow*, Boolean *doAutoSurveyHigh*)
Sets whether to do auto-survey on the high and low intensity limits.
¬†
void **LinePlotImageDisplaySetFilled**(LinePlotImageDisplay *lpid*, Boolean *on*)
Sets whether to fill the lineplot.
¬†
void **LinePlotImageDisplaySetFrameOn**(LinePlotImageDisplay *lpid*, Boolean *on*)
Sets whether to draw the frame.
¬†
void **LinePlotImageDisplaySetGridOn**(LinePlotImageDisplay *lpid*, Boolean *on*)
Sets whether to draw the grid.
¬†
void **LinePlotImageDisplaySetSlice**(LinePlotImageDisplay *lpid*, Number *slice*)
Sets the slice currently displayed at the bottom.
¬†
void **LinePlotImageDisplaySetSliceComponentColor**(LinePlotImageDisplay *lpid*, Number *slice_index*, Number *comp_index*, Number *r*, Number *g*, Number *b*)
Sets the color of the 'comp_index'th component of the 'slice_index'th slice.
¬†
void **LinePlotImageDisplaySetSliceDrawingStyle**(LinePlotImageDisplay *lpid*, Number *slice_index*, Number *style*)
Sets the drawing style of the 'slice_index'th slice.
¬†
void **LinePlotImageDisplaySetTrackingStyle**(LinePlotImageDisplay *lpid*, Number *track_style_x*, Number *track_style_y*)
Sets the tracking style of the line plot.
¬†
void **LoadNotes**(ImageReference, String *tagPath*, String *fileName*)
Loads the image notes from the given fileName into tagPath. This function will read the group name from the file. The tagPath must refer to a group.
¬†
void **LoadNotesAs**(ImageReference, String *tagPath*, String *fileName*)
Loads the image notes from the given fileName into tagPath. The tagPath must refer to a group.
¬†
void **LoadPersistentNotes**(String *tagPath*, String *fileName*)
Loads the persistent notes from the given fileName into tagPath. This function will read the group name from the file. The tagPath must refer to a group.
¬†
void **LoadPersistentNotesAs**(String *tagPath*, String *fileName*)
Loads the persistent notes from the given fileName into tagPath. The tagPath must refer to a group.
¬†
Number **log**(Number *v1*)
¬†
ComplexNumberExpression **log**(ComplexNumberExpression)
Return the logarithm of the number.
¬†
RealNumberExpression **log**(RealNumberExpression)
Return the natural logarithm of the number.
¬†
Number **log1**(Number *v1*)
¬†
RealNumberExpression **log1**(RealNumberExpression)
Return the natural logarithm of the number + 1.
¬†

Number **log10**(Number *v1*)
¬†
RealNumberExpression **log10**(RealNumberExpression)
Return the logarithm base 10 of the number.
¬†
Number **log2**(Number *v1*)
¬†
RealNumberExpression **log2**(RealNumberExpression)
Return the logarithm base 2 of the number.
¬†
RealNumberExpression **LogGamma**(Number)
Return the log gamma of the number.
¬†
RealImage **LUDecomposition**(RealImage *a*, RealImage *b*)
Return the image resulting from a LU decomposition on images a,b.
¬†
Number **MatrixDeterminant**(RealImage *a*)
Return the matrix determinant number of matrix image a.
¬†
RealImage **MatrixInverse**(RealImage *a*)
Return the matrix inverse image of matrix image a.
¬†
RealImage **MatrixMultiply**(RealImage *a*, RealImage *b*)
Return the matrix product image of matrix images a and b.
¬†
void **MatrixPrint**(ImageReference *a*)
Print an image as a matrix to the results window.
¬†
ComplexImage **MatrixTranspose**(ComplexImage *a*)
Return the matrix transpose image of matrix image a.
¬†
RGBImage **MatrixTranspose**(RGBImage *a*)
Return the matrix transpose image of matrix image a.
¬†
RealImage **MatrixTranspose**(RealImage *a*)
Return the matrix transpose image of matrix image a.
¬†
RealNumberExpression **max**(RealNumberExpression, RealNumberExpression)
Return the maximum of the two numbers.
¬†
Number **max**(RealImageExpression, NumberVariable *x*, NumberVariable *y*)
Finds the maximum pixel in the image, stores the coordinates of that pixel into x and y, and returns the value of the pixel.
¬†
Number **max**(RealImageExpression)
Return the maxmimum pixel within the image expression.
¬†
Number **max**(Number *v1*, Number *v2*)
¬†
RealNumberExpression **Maximum**(RealNumberExpression...)
Return the maximum of a list of numbers.
¬†
Number **mean**(RealImageExpression)
Return the mean pixel value of the image expression.
¬†
Number **MeanSquare**(RealImageExpression)
Return the mean square of all pixel values of the image expression.
¬†
RealNumberExpression **Median**(RealNumberExpression...)
Return the median of a list of numbers.
¬†
RealImage **MedianFilter**(ImageReference *source*, Number *filterType*, Number *size*)
Performs a median filter on the source image according to the filterType parameter (0=horizontal, 1=vertical, 2=cross, 3=entire) and the size parameter. Size specifies size in each direction - so a 'size' of 2 is a 5x5 window.
¬†
String **mid**(String, Number *offset*, Number *count*)

Returns count characters of a string starting at offset.
¬†
`Number` **`min`**`( Number` *`v1`*`, Number` *`v2`*` )`
¬†
`RealNumberExpression` **`min`**`( RealNumberExpression, RealNumberExpression )`
Return the minimum of the two numbers.
¬†
`Number` **`min`**`( RealImageExpression, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Finds the minimum pixel in the image, stores the coordinates of that pixel into x and y, and returns the value of the pixel.
¬†
`Number` **`min`**`( RealImageExpression )`
Return the minimum pixel within the image expression.
¬†
`RealNumberExpression` **`Minimum`**`( RealNumberExpression... )`
Return the minimum of a list of numbers.
¬†
`void` **`minmax`**`( RealImageExpression, NumberVariable` *`minP`*`, NumberVariable` *`maxP`*` )`
Finds the minimum and maximum pixel in the image and store those values into minP and maxP.
¬†
`RealNumberExpression` **`mod`**`( RealNumberExpression, RealNumberExpression )`
Return the modulus of the first number divided by the second number.
¬†
`Number` **`mod`**`( Number` *`v1`*`, Number` *`v2`*` )`
¬†
`void` **`ModelessDialog`**`( String` *`prompt`*`, String` *`buttonName`*`, Number` *`semaphore`*` )`
Present a modeless dialog with the prompt and buttonName. When the user presses the button, the semaphore will be cleared. This function can only be used in the background.
¬†
`RealNumberExpression` **`modulus`**`( ComplexNumberExpression )`
Return the modulus of a complex number.
¬†
`void` **`MoveAnnotation`**`( ImageReference, Number` *`annotationID`*`, Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`*` )`
Set the bounds of the annotation indicated by the annotationID within the image to [top, left, bottom, right].
¬†
`RealImage` **`MPClose`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically close the image using the neighbors parameter to control the closing and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPDilate`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically dilate the image using the neighbors parameter to control the dilation and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPDistanceMap`**`( ImageReference` *`image`*` )`
Generate a distance map from the source image and return the resulting real image. The source image must be binary.
¬†
`RealImage` **`MPErode`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically erode the image using the neighbors parameter to control the erosion and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPEuclideanDistanceMap`**`( ImageReference` *`image`*` )`
Generate a Euclidean distance map from the source image and return the resulting real image. The source image must be binary.
¬†
`ComplexImage` **`MPExactDistanceMap`**`( ImageReference` *`image`*` )`
Generate an exact distance map from the source image and return the resulting complex image. The source image must be binary.
¬†
`RealImage` **`MPOpen`**`( RealImage` *`image`*`, Number` *`neighbors`*` )`
Morphologically open the image using the neighbors parameter to control the opening and return the resulting image. The source image must be binary.
¬†
`RealImage` **`MPOutline`**`( RealImage` *`image`*` )`
Morphologically outline the image and return the resulting image. The source image must be binary.
¬†
`Number` **`Nan`**`( String` *`tag`*`, Boolean` *`is_signaling`*` )`

Returns a nan containing the specified tag.
¬†
`Number` **`nearest`**`( Number` *`v1`* `)`
¬†
`RealNumberExpression` **`Nearest`**`( RealNumberExpression )`
Return the number rounded to the nearest integer.
¬†
`Function` **`NewAbstractMethod`**`( String` *`method_name`*`, String` *`method_signature`* `)`
Returns an abstract method with name 'method_name' and signature 'method_signature'.
¬†
`Component` **`NewArrowAnnotation`**`( Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Creates a new arrow annotation.
¬†
`Component` **`NewBoxAnnotation`**`( Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Creates a new box annotation.
¬†
`Function` **`NewCallbackFunction`**`( String` *`method_signature`*`, ! callback, Number` *`linkage_style`* `)`
Creates a function representing a callback to a C function 'callback'.
¬†
`Component` **`NewComponent`**`( Number` *`type`*`, Number` *`f1`*`, Number` *`f2`*`, Number` *`f3`*`, Number` *`f4`* `)`
Creates a new annotaiton of type 'type'
¬†
`Component` **`NewDoubleArrowAnnotation`**`( Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Creates a new double arrow annotation.
¬†
`Function` **`NewFunctionFromScript`**`( String` *`script`*`, String` *`signature`* `)`
Compiles script and returns the function matching signature.
¬†
`Component` **`NewGroupAnnotation`**`( )`
Creates a new group annotation.
¬†
`ImageReference` **`NewImage`**`( String` *`title`*`, Number` *`type`*`, Number` *`d0`*`, Number` *`d1`*`, Number` *`d2`*`, Number` *`d3`* `)`
Creates a 4D image of type 'type' and size [d0,d1,d2,d3] with the given title.
¬†
`ImageReference` **`NewImage`**`( String` *`title`*`, Number` *`type`*`, Number` *`d0`*`, Number` *`d1`*`, Number` *`d2`* `)`
Creates a 3D image of type 'type' and size [d0,d1,d2] with the given title.
¬†
`ImageReference` **`NewImage`**`( String` *`title`*`, Number` *`type`*`, Number` *`d0`*`, Number` *`d1`* `)`
Creates a 2D image of type 'type' and size [d0,d1] with the given title.
¬†
`ImageReference` **`NewImage`**`( String` *`title`*`, Number` *`type`*`, Number` *`d0`* `)`
Creates a 1D image of type 'type' and size [d0] with the given title.
¬†
`ImageDocument` **`NewImageDocument`**`( String` *`title`* `)`
Creates an empty image document.
¬†
`ImageDocument` **`NewImageDocumentFromFile`**`( String` *`path_name`* `)`
Creates a new image document from a file.
¬†
`ImageReference` **`NewImageFromFile`**`( String` *`file_path`* `)`
Opens a file and reads it as an image.
¬†
`Component` **`NewLineAnnotation`**`( Number` *`top`*`, Number` *`left`*`, Number` *`bottom`*`, Number` *`right`* `)`
Creates a new line annotation.
¬†
`ImageReference` **`NewLiveFFT`**`( ImageDisplay` *`imageDisplay`*`, ROI` *`roi`*`, Boolean` *`reduce`* `)`
Creates a new live fft of the area in 'roi', that is reduced if 'reduce' is 'true'
¬†
`ImageReference` **`NewLiveHistogram`**`( ImageDisplay` *`imageDisplay`*`, ROI` *`roi`*`, Number` *`num_channels`* `)`
Creates a new live histogram of the area in 'roi', binned by 'num_channels'
¬†
`ImageReference` **`NewLiveProfile`**`( ImageDisplay` *`imageDisplay`*`, Number` *`start_x`*`, Number` *`start_y`*`, Number` *`end_x`*`, Number` *`end_y`*`, Number` *`width`* `)`
Creates a new live profile from (start_x,start_y) to (end_x,end_y)
¬†

Component **NewOvalAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new oval annotation.
¬†
Component **NewPictureAnnotation**(Number *top*, Number *left*, Number *bottom*, Number *right*, Number *picture*)
Creates a new picture annotation.
¬†
ROI **NewROI**( )
Creates an empty region of interest.
¬†
DocumentWindow **NewScriptWindow**(String *title*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Creates a new editor window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*, String *font_name*, Number *attributes*, Number *size*, Number *encoding*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Opens a file into a script window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*)
Opens a file into a script window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Opens a file into a script window.
¬†
DocumentWindow **NewScriptWindowFromFile**(String *file_name*, String *font_name*, Number *attributes*, Number *size*, Number *encoding*)
Opens a file into a script window.
¬†
Number **NewSemaphore**( )
Create a semaphore. Used only with background processing.
¬†
ScriptObject **NewStreamFromBuffer**(Number *initial_size*)
Creates a new stream object from from a memory buffer with initial size 'initial_size'.
¬†
ScriptObject **NewStreamFromFileReference**(Number *file_ref*, Boolean *do_close*)
Creates a new stream object from the file reference. On destruction, closes the file reference if 'do_close' is true.
¬†
TagGroup **NewTagGroup**( )
Creates an empty tag group.
¬†
TagGroup **NewTagList**( )
Creates an empty tag list.
¬†
Component **NewTextAnnotation**(Number *left*, Number *top*, String *text*, Number *size*)
Creates a new text annotation.
¬†
ImageReference **NextImage**(ImageReference)
Find the next image.
¬†
RGBImageExpression **NoClipIndex**(RGBImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)
Returns a pixel in the given RGB image at the position [x,y,z]. Does not perform bounds checking. This is slightly faster than Index().
¬†
RealImageExpression **NoClipIndex**(RealImage, RealImageExpression *x*, RealImageExpression *y*)
Returns a pixel in the given real image at the position [x,y]. Does not perform bounds checking. This is slightly faster than Index().
¬†
RGBImageExpression **NoClipIndex**(RGBImage, RealImageExpression *x*, RealImageExpression *y*)
Returns a pixel in the given RGB image at the position [x,y]. Does not perform bounds checking. This is slightly faster than Index().
¬†
ComplexImageExpression **NoClipIndex**(ComplexImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)
Returns a pixel in the given Complex image at the position [x,y,z]. Does not perform bounds checking. This is slightly faster than Index().
¬†
RealImageExpression **NoClipIndex**(RealImage, RealImageExpression *x*, RealImageExpression *y*, RealImageExpression *z*)

Returns a pixel in the given real image at the position [x,y,z]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`ComplexImageExpression` **`NoClipIndex`**`( ComplexImage, RealImageExpression x, RealImageExpression y )`
Returns a pixel in the given Complex image at the position [x,y]. Does not perform bounds checking. This is slightly faster than Index().
¬†

`RealNumberExpression` **`norm`**`( ComplexNumberExpression )`
Return the norm of a complex number. This is the modulus squared.
¬†

`void` **`ObjectTransformCompose`**`( Number i1_o_x, Number i1_o_y, Number i1_s_x, Number i1_s_y, Number i2_o_x, Number i2_o_y, Number i2_s_x, Number i2_s_y, NumberVariable o_o_x, NumberVariable o_o_y, NumberVariable o_s_x, NumberVariable o_s_y )`
Composes the transform (i1_o_x,i1_o_y,i1_s_x,i1_s_y) with (i2_o_x,i2_o_y,i2_s_x,i2_s_y) and places the result in (*o_o_x,*o_o_y,*o_s_x,*o_s_y).
¬†

`void` **`ObjectTransformInvert`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, NumberVariable o_o_x, NumberVariable o_o_y, NumberVariable o_s_x, NumberVariable o_s_y )`
Inverts the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_o_x,*o_o_y,*o_s_x,*o_s_y).
¬†

`void` **`ObjectTransformTransformPoint`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_p_x, Number i_p_y, NumberVariable o_p_x, NumberVariable o_p_y )`
Transforms the point (i_p_x,i_p_y) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_p_x,*o_p_y).
¬†

`void` **`ObjectTransformTransformRect`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_r_t, Number i_r_l, Number i_r_b, Number i_r_r, NumberVariable o_r_t, NumberVariable o_r_l, NumberVariable o_r_b, NumberVariable o_r_r )`
Transforms the point (i_r_t,i_r_l,i_r_b,i_r_r) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_r_t,*o_r_l,*o_r_b,*o_r_r).
¬†

`void` **`ObjectTransformUntransformPoint`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_p_x, Number i_p_y, NumberVariable o_p_x, NumberVariable o_p_y )`
Transforms the point (i_p_x,i_p_y) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_p_x,*o_p_y).
¬†

`void` **`ObjectTransformUntransformRect`**`( Number i_o_x, Number i_o_y, Number i_s_x, Number i_s_y, Number i_r_t, Number i_r_l, Number i_r_b, Number i_r_r, NumberVariable o_r_t, NumberVariable o_r_l, NumberVariable o_r_b, NumberVariable o_r_r )`
Transforms the point (i_r_t,i_r_l,i_r_b,i_r_r) by the transform (i_o_x,i_o_y,i_s_x,i_s_y) and places the result in (*o_r_t,*o_r_l,*o_r_b,*o_r_r).
¬†

`String` **`Octal`**`( Number n )`
Returns the number as an octal string.
¬†

`String` **`Octal`**`( Number n, Number length )`
Returns the number the number as an octal string of the given length.
¬†

`ComplexImageExpression` **`Offset`**`( ComplexImage, Number deltax, Number deltay )`
Returns the pixel with the offset specified by [dh,dv] from the current evaluation position within the image.
¬†

`RGBImageExpression` **`Offset`**`( RGBImage, Number deltax, Number deltay )`
Returns the pixel with the offset specified by [dh,dv] from the current evaluation position within the image.
¬†

`RealImageExpression` **`Offset`**`( RealImage, Number dh, Number dv )`
Returns the pixel with the offset specified by [dh,dv] from the current evaluation position within the image.
¬†

`void` **`OffsetAnnotation`**`( ImageReference, Number annotationID, Number deltax, Number deltay )`
Offset the annotation indicated by the annotationID within the image by [deltax, deltay].
¬†

`Boolean` **`OkCancelDialog`**`( String prompt )`
Puts up a dialog with the given prompt. Returns 1 for OK and 0 for Cancel.
¬†

`void` **`OkDialog`**`( String prompt )`
Puts up a dialog with the given prompt.
¬†

`void` **`OpenAndSetProgressWindow`**`( String line1, String line2, String line3 )`
Open the progress window and sets the text within to line1, line2, and line3.
¬†

`Boolean` **`OpenDialog`**`( String pathname )`

Puts up an Open dialog, allows the user to select a file, and stores the pathname into the pathname variable. Returns 1 for OK and 0 for Cancel.
¬†

`Number` **`OpenFileForReading`**`( String` *`fileName`* `)`
Open the file for reading. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.
¬†

`Number` **`OpenFileForReadingAndWriting`**`( String` *`fileName`* `)`
Open the file for reading and writing. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.
¬†

`Number` **`OpenFileForWriting`**`( String` *`fileName`* `)`
Open the file for writing. Return the file reference for this file. This call must be balanced with call to CloseFile() with the returned reference number.
¬†

`ImageReference` **`OpenImage`**`( String` *`fileName`* `)`
Open the image with the filename. Returns the opened image.
¬†

`void` **`OpenResultsWindow`**`( )`
Open the results window if it is not already open.
¬†

`void` **`OpenTimeBar`**`( String` *`prompt`*`, Number` *`total`* `)`
Opens the time bar with the given prompt. The total parameter is ignored. CloseTimeBar() must be invoked exactly once for every OpenTimeBar() call.
¬†

`Boolean` **`OptionDown`**`( )`
Returns 1 if the option key is down and 0 otherwise.
¬†

`void` **`OutputCode`**`( Number` *`x`* `)`
¬†

`void` **`OutputDAGs`**`( Number` *`x`* `)`
¬†

`void` **`OutputTrees`**`( Number` *`x`* `)`
¬†

`ComplexImage` **`PackedFFT`**`( RealImage` *`source`* `)`
Creates a new packed complex image from the FFT of the real image source.
¬†

`RealImage` **`PackedIFFT`**`( ComplexImage` *`source`* `)`
Creates a new real image from the inverse FFT of the packed complex image source.
¬†

`ComplexImage` **`PackedToComplex`**`( ComplexImage` *`source`* `)`
Creates a new complex 8-byte image from the packed complex source.
¬†

`String` **`PathAddParentIndirection`**`( String` *`path`* `)`
Returns 'path' appended with a string denoting indirection to the parent directory
¬†

`String` **`PathBeginRelative`**`( )`
Returns a string that begins a relative path
¬†

`String` **`PathConcatenate`**`( String` *`initial_path`*`, String` *`final_path`* `)`
Concatenates 'final_path' to 'initial_path' to create a new path, adding separators as necessary.
¬†

`String` **`PathExtractBaseName`**`( String` *`path`*`, Number` *`path_type`* `)`
Returns the base name portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

`String` **`PathExtractDirectory`**`( String` *`path`*`, Number` *`path_type`* `)`
Returns the directory portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

`String` **`PathExtractExtension`**`( String` *`path`*`, Number` *`path_type`* `)`
Returns the extension portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

`String` **`PathExtractFileName`**`( String` *`path`*`, Number` *`path_type`* `)`
Returns the file name portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

String **PathExtractParentDirectory**(String *path*, Number *path_type*)
Returns the parent directory portion of 'dir_path', where 'path_type' denotes '1' for Mac paths, '2' for Windows paths, and '0' for current OS paths
¬†

String **PathGetFullpath**(String *path*)
Returns the full path name of the file denoted by 'dir_path'
¬†

RealNumberExpression **Phase**(ComplexNumberExpression)
Return the phase of the complex number.
¬†

Number **Pi**( )
Return an approximation of pi.
¬†

void **PictureAnnotationSetPicture**(Component *comp*, Number *picture*)
Sets the picture of an annotation.
¬†

void **PIctureGetBounds**(Number *picture*, NumberVariable *top*, NumberVariable *left*, NumberVariable *bottom*, NumberVariable *right*)
Gets the preferred bounds of the picture for display on the screen.
¬†

RealNumberExpression **PoissonRandom**(Number)
Return a random number with poisson distribution between [0,1).
¬†

ComplexNumberExpression **Polar**(ComplexNumberExpression)
Return the polar representation of the rectangular complex number.
¬†

ComplexNumber **Polynomial**(ComplexImageExpression, ComplexNumber *x*)
Return the polynomial a0 + a1*x + a2*x*x ... where x is the free variable and a0 ... aN are the pixels of the image.
¬†

Number **Polynomial**(RealImageExpression, Number *x*)
Return the polynomial a0 + a1*x + a2*x*x ... where x is the free variable and a0 ... aN are the pixels of the image.
¬†

void **print**(Number *x*)
¬†

void **PrintImage**(ImageReference)
Print the image.
¬†

Number **product**(RealImageExpression)
Return the product of all pixel values of the image expression.
¬†

ComplexNumber **product**(ComplexImageExpression)
Return the product of all pixel values of the image expression.
¬†

RealNumberExpression **random**( )
Return a random number in the range 0 to 65535.
¬†

void **RasterImageDisplayAddThresholdToMask**(RasterImageDisplay *rid*, ImageReference *mask*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the points in mask to 1 if they lie within the threshold.
¬†

void **RasterImageDisplayGetThresholdLimits**(RasterImageDisplay *rid*, NumberVariable *low*, NumberVariable *high*)
Gets the threshold limits of the display.
¬†

Boolean **RasterImageDisplayIsThresholdOn**(RasterImageDisplay *rid*)
Determines whether the thresholding overlay is on or off.
¬†

RasterImageDisplay **RasterImageDisplayNullify**(!)
¬†

void **RasterImageDisplaySetThresholdLimits**(RasterImageDisplay *rid*, Number *low*, Number *high*)
Sets the threshold limits of the display.
¬†

void **RasterImageDisplaySetThresholdOn**(RasterImageDisplay *rid*, Boolean *on*)
Sets whether the thresholding overlay is on or off.
¬†

RealImage **RasterizeRGB**(RGBImage *source*, Boolean *dither*)
Rasterize the source RGB image and return the resulting Raster image displayed image with an appropriate color table. The dither parameter controls dithering.
¬†

void **RawCopyImage**(ImageReference *src*, ImageReference *dst*)
Copies the src image to the dst image ignoring data types. The data type sizes of the two images must be the same.
¬†

String **ReadFile**(Number *file*, Number *encoding*, Number *count*)
Reads count bytes from a file, returning them as a string assuming they have the specified encoding.
¬†

String **ReadFile**(Number *file*, Number *count*)
Read count bytes from the file, returning them as a string.
¬†

Boolean **ReadFileLine**(Number *file*, String *string*)
Read a line of text from the file, storing it into the string variable. Return 1 if successful and 0 otherwise.
¬†

Boolean **ReadFileLine**(Number *file*, Number *encoding*, String *string*)
Read a line of text from the file, storing it into the string variable, assuming the text has the specified encoding. Return 1 if successful and 0 otherwise.
¬†

void **ReadRawStream**(Number *rawStream*, Number *data*, Number *length*)
Read length bytes from rawStream and store it into the memory pointed to by data.
¬†

RealNumberExpression **real**(ComplexNumberExpression)
Return the real portion of the complex number.
¬†

ComplexImage **RealFFT**(RealImage *source*)
Creates a new complex 8-byte image from the FFT of the real image source.
¬†

RealImage **RealIFFT**(ComplexImage *source*)
Creates a new real image from the inverse FFT of the complex 8-byte image source.
¬†

RealImage **RealImage**(String *title*, Number *bytes*, Number *d0*)
Creates a 1D real image of size [d0] with the given title. The bytes parameter can be 4 or 8 for single and double precision floating point numbers.
¬†

RealImage **RealImage**(String *title*, Number *bytes*, Number *d0*, Number *d1*)
Creates a 2D real image of size [d0,d1] with the given title. The bytes parameter can be 4 or 8 for single and double precision floating point numbers.
¬†

RealImage **RealImage**(String *title*, Number *bytes*, Number *d0*, Number *d1*, Number *d2*)
Creates a 3D real image of size [d0,d1,d2] with the given title. The bytes parameter can be 4 or 8 for single and double precision floating point numbers.
¬†

ComplexNumberExpression **Rect**(ComplexNumberExpression)
Return the rectangular representation of the polar complex number.
¬†

RealNumberExpression **red**(RGBNumberExpression)
Return the red portion of an RGB number.
¬†

void **Reduce**(ImageReference)
Reduces the image by 2X.
¬†

ComplexImage **ReducedFFT**(ImageReference *source*)
Creates a new packed complex image from the FFT of the real image source after reducing the source by a factor of 2.
¬†

void **RegisterClass**(String *class_name*, String *parent_class_name*, ! *alloc_proc*, ! *dealloc_proc*, Number *refCon*)
¬†

Number **RegisterCustomMenu**(Number *menuHandler*)
Register a custom menu. See the SDK documentation for more information.
¬†

void **RegisterMenuAdjustment**(String *menuName*, String *adjustedMenuName*)
Unregister a menu name adjustment. See the SDK documentation for more information.

¬†

Number **RegisterObjectListener**(Number *object*, Number *proc*, Number *refCon*)

Add object listener to OM object. See the SDK documentation for more information.

¬†

void **RegisterScriptPalette**(ScriptObject, String *type*, String *name*)

¬†

Number **ReinterpretComponentAsLong**(Component *comp*)

¬†

Number **ReinterpretImageAsLong**(ImageReference *img*)

¬†

Number **ReinterpretImageDisplayAsLong**(ImageDisplay *imgDisp*)

¬†

Number **ReinterpretImageDocumentAsLong**(ImageDocument *id*)

¬†

Component **ReinterpretLongAsComponent**(Number *l*)

¬†

Component **ReinterpretLongAsComponentPtr**(Number *l*)

¬†

ImageReference **ReinterpretLongAsImage**(Number *l*)

¬†

ImageDisplay **ReinterpretLongAsImageDisplay**(Number *l*)

¬†

ImageDisplay **ReinterpretLongAsImageDisplayPtr**(Number *l*)

¬†

ImageDocument **ReinterpretLongAsImageDocument**(Number *l*)

¬†

ImageDocument **ReinterpretLongAsImageDocumentPtr**(Number *l*)

¬†

ImageVariable **ReinterpretLongAsImagePtr**(Number *l*)

¬†

ROI **ReinterpretLongAsRegionOfInterest**(Number *l*)

¬†

ROI **ReinterpretLongAsROI**(Number *l*)

¬†

ROI **ReinterpretLongAsROIPtr**(Number *l*)

¬†

ScriptObject **ReinterpretLongAsScriptObject**(Number *l*)

¬†

TagGroup **ReinterpretLongAsTagGroup**(Number *l*)

¬†

TagGroup **ReinterpretLongAsTagGroupPtr**(Number *l*)

¬†

DocumentWindow **ReinterpretLongAsWindow**(Number *l*)

¬†

DocumentWindow **ReinterpretLongAsWindowPtr**(Number *l*)

¬†

Number **ReinterpretRegionOfInterestAsLong**(ROI *tg*)

¬†

Number **ReinterpretROIAsLong**(ROI *tg*)

¬†

Number **ReinterpretScriptObjectAsLong**(ScriptObject *tg*)

¬†

Number **ReinterpretTagGroupAsLong**(TagGroup *tg*)

¬†

Number **ReinterpretWindowAsLong**(DocumentWindow *id*)

¬†

void **ReleaseSemaphore**(Number)

Release a semaphore. Used only with background processing.

¬†

RealNumberExpression **remainder**(RealNumberExpression, RealNumberExpression)

Return the remainder of the first number divided by the second number.

¬†

void **RemoveDaemonHack**(ImageReference, Number *id*)

Remove the daemon component for the image. See the SDK documentation for more information.

¬†

void **RemovePathFromCopyToImageList**(String *path*)

Removes 'path' from the copy to image list.
¬†

`void` **RemoveScriptFromMenu**`( String` *commandName*`, String` *menuName*`, String` *optionalSubMenuName* `)`
Removes the given menu command from the menu. The commandName indicates the string by which this script is known to the application. The menuName and optionalSubMenuName parameters specify the menu.
¬†

`void` **Result**`( String )`
Output the string to the results window.
¬†

`void` **Result**`( Number, String` *format* `)`
Output the real number with the printf-style format to the results window.
¬†

`void` **Result**`( Number )`
Output the real number to the results window.
¬†

`void` **Result**`( RGBNumber )`
Output the RGB number to the results window.
¬†

`void` **Result**`( ComplexNumber )`
Output the complex number to the results window.
¬†

`void` **return**`( Expression )`
¬†

`void` **return**`( )`
¬†

`RGBNumberExpression` **rgb**`( RealNumberExpression` *red*`, RealNumberExpression` *green*`, RealNumberExpression` *blue* `)`
Creates an RGB number from the individual number components.
¬†

`RGBNumberExpression` **rgba**`( RealNumberExpression` *red*`, RealNumberExpression` *green*`, RealNumberExpression` *blue*`, RealNumberExpression` *alpha* `)`
Creates an RGB number from the individual number components, including the alpha channel.
¬†

`RGBImage` **RGBImage**`( String` *title*`, Number` *bytes*`, Number` *d0*`, Number` *d1*`, Number` *d2* `)`
Creates a 3D RGB image of size [d0,d1,d2] with the given title. The bytes parameter must be 4.
¬†

`RGBImage` **RGBImage**`( String` *title*`, Number` *bytes*`, Number` *d0*`, Number` *d1* `)`
Creates a 2D RGB image of size [d0,d1] with the given title. The bytes parameter must be 4.
¬†

`RGBImage` **RGBImage**`( String` *title*`, Number` *bytes*`, Number` *d0* `)`
Creates a 1D RGB image of size [d0] with the given title. The bytes parameter must be 4.
¬†

`String` **right**`( String, Number` *count* `)`
Returns rightmost count characters of a string.
¬†

`Number` **RMS**`( RealImageExpression )`
Return the RMS of all pixel values of the image expression.
¬†

`void` **ROIAddToMask**`( ROI` *roi*`, ImageReference` *mask*`, Number` *top*`, Number` *left*`, Number` *bottom*`, Number` *right* `)`
Add the region of interest to the image within the bounds of the specified rectangle.
¬†

`void` **ROIAddVertex**`( ROI` *roi*`, Number` *x*`, Number` *y* `)`
Add a vertex with the given coordinates to the region of interest.
¬†

`void` **ROIClearVertices**`( ROI` *roi* `)`
Remove all vertices from the region of interest.
¬†

`ROI` **ROIClone**`( ROI` *roi* `)`
Returns a clone of the roi.
¬†

`Boolean` **ROIContainsPoint**`( ROI` *roi*`, Number` *x*`, Number` *y* `)`
Returns whether the region of interest encloses the given point.
¬†

`Number` **ROICountVertices**`( ROI` *roi* `)`

Return the number of vertices comprising the region of interest.
¬†
`void` **`ROIDeleteVertex`**`( ROI` *`roi`*`, Number` *`index`* `)`
Delete the given vertex from the region of interest.
¬†
`void` **`ROIGetColor`**`( ROI` *`roi`*`, NumberVariable` *`r`*`, NumberVariable` *`g`*`, NumberVariable` *`b`* `)`
Stores the color of the region of interest into the variables. Each number will be in the range of 0 to 1.
¬†
`Boolean` **`ROIGetDeletable`**`( ROI` *`roi`* `)`
Return whether the region of interest is deletable or not.
¬†
`Number` **`ROIGetID`**`( ROI` *`roi`* `)`
Return the ID for the region of interest.
¬†
`String` **`ROIGetLabel`**`( ROI` *`roi`* `)`
Return the label of the region of interest.
¬†
`void` **`ROIGetLine`**`( ROI` *`roi`*`, NumberVariable` *`sx`*`, NumberVariable` *`sy`*`, NumberVariable` *`ex`*`, NumberVariable` *`ey`* `)`
Fill in the start and end points of the line represented by the region of interest.
¬†
`Boolean` **`ROIGetMoveable`**`( ROI` *`roi`* `)`
Return whether the region of interest is moveable or not.
¬†
`String` **`ROIGetName`**`( ROI` *`roi`* `)`
Return the name of the region of interest.
¬†
`void` **`ROIGetPoint`**`( ROI` *`roi`*`, NumberVariable` *`x`*`, NumberVariable` *`y`* `)`
Return the coordinates of the point represented by this region of interest.
¬†
`void` **`ROIGetRange`**`( ROI` *`roi`*`, NumberVariable` *`start`*`, NumberVariable` *`end`* `)`
Fills in the start and end columns of the range represented by the region of interest.
¬†
`void` **`ROIGetRectangle`**`( ROI` *`roi`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`
Fill in the coordinates of the rectangle represented by the region of interest.
¬†
`Boolean` **`ROIGetResizable`**`( ROI` *`roi`* `)`
Return whether the region of interest is resizable or not.
¬†
`void` **`ROIGetVertex`**`( ROI` *`roi`*`, Number` *`index`*`, NumberVariable` *`x`*`, NumberVariable` *`y`* `)`
Return the coordinates of the given vertex of the region of interest.
¬†
`Boolean` **`ROIGetVolatile`**`( ROI` *`roi`* `)`
Return whether the region of interest is volatile or not.
¬†
`void` **`ROIInsertVertex`**`( ROI` *`roi`*`, Number` *`before`*`, Number` *`x`*`, Number` *`y`* `)`
Insert a vertex with the given coordinates before the indicated vertex of the region of interest.
¬†
`Boolean` **`ROIIsClosed`**`( ROI` *`roi`* `)`
Returns whether the region of interest is a closed loop or not.
¬†
`Boolean` **`ROIIsLine`**`( ROI` *`roi`* `)`
Return whether the region of interest is a line.
¬†
`Boolean` **`ROIIsPoint`**`( ROI` *`roi`* `)`
Return whether the region of interest is a point.
¬†
`Boolean` **`ROIIsRange`**`( ROI` *`roi`* `)`
Returns whether the region of interest is a range.
¬†
`Boolean` **`ROIIsRectangle`**`( ROI` *`roi`* `)`
Return whether the region of interest is a rectangle.
¬†
`Boolean` **`ROIIsValid`**`( ROI` *`roi`* `)`

Returns 'true' if the region of interest is a valid object.
¬†
ROI **ROINullify**(!)
¬†
void **ROISetColor**(ROI *roi*, Number *r*, Number *g*, Number *b*)
Set the color of the region of interest. Each number should be in the range of 0 to 1.
¬†
void **ROISetDeletable**(ROI *roi*, Boolean *deletable*)
Sets whether the region of interest should be deletable or not.
¬†
void **ROISetIsClosed**(ROI *roi*, Boolean *is_closed*)
Sets whether the region of interest is a closed loop or not (that is the last vertex connects to the first).
¬†
void **ROISetLabel**(ROI *roi*, String *name*)
Set the label on the region of interest.
¬†
void **ROISetLine**(ROI *roi*, Number *sx*, Number *sy*, Number *ex*, Number *ey*)
Set the region of interest to a line with the given start and end coordinates.
¬†
void **ROISetMoveable**(ROI *roi*, Boolean *moveable*)
Sets whether the region of interest should be moveable or not.
¬†
void **ROISetName**(ROI *roi*, String *name*)
Set the name of the region of interest.
¬†
void **ROISetPoint**(ROI *roi*, Number *x*, Number *y*)
Set the region of interest to a point with the given coordinate.
¬†
void **ROISetRange**(ROI *roi*, Number *start*, Number *end*)
Sets the region of interest to a range with the given start and end columns.
¬†
void **ROISetRectangle**(ROI *roi*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Set the region of interest to a rectangle with the given coordinates.
¬†
void **ROISetRegionToValue**(ROI *roi*, ComplexImage *mask*, ComplexNumber *value*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the area in 'mask' corresponding to the region to the value 'value'.
¬†
void **ROISetRegionToValue**(ROI *roi*, RGBImage *mask*, RGBNumber *value*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the area in 'mask' corresponding to the region to the value 'value'.
¬†
void **ROISetRegionToValue**(ROI *roi*, ImageReference *mask*, Number *value*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the area in 'mask' corresponding to the region to the value 'value'.
¬†
void **ROISetResizable**(ROI *roi*, Boolean *resizable*)
Sets whether the region of interest should be resizable or not.
¬†
void **ROISetVertex**(ROI *roi*, Number *index*, Number *x*, Number *y*)
Set the coordinates of the given vertex of the region of interest.
¬†
void **ROISetVolatile**(ROI *roi*, Boolean *is_volatile*)
Set whether the region of interest is volatile or not.
¬†
RealImage **Rotate**(ImageReference *source*, Number *radians*)
Creates a new real image by rotating the source image counterclockwise by radians.
¬†
void **RotateLeft**(ImageReference)
Rotates the image to the left by 90¬°.
¬†
void **RotateRight**(ImageReference)
Rotates the image to the right by 90¬°.
¬†
Number **round**(Number *v1*)

¬†

`RealNumberExpression` **`Round`**`( RealNumberExpression )`
Return the number converted to integer using current rounding mode.

¬†

`void` **`Save`**`( ImageReference )`
Save the image under the it's current filename.

¬†

`Boolean` **`SaveAsDialog`**`( String` *`prompt`*`, String` *`defaultName`*`, String` *`saveName`*` )`
Puts up the SaveAs dialog with the given prompt and default file name, and then stores the chosen path for the saved file into the saveName variable. Returns 1 for OK and 0 for Cancel.

¬†

`void` **`SaveAsGatan`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in Gatan 3.0 file format.

¬†

`void` **`SaveAsGIF`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in GIF file format. The DM Import/Export Plug-in must be installed in order for this function to work.

¬†

`void` **`SaveAsPCX`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in PCX file format. The DM Import/Export Plug-in must be installed in order for this function to work.

¬†

`void` **`SaveAsPICT`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in PICT file format. The DM Import/Export Plug-in must be installed in order for this function to work.

¬†

`void` **`SaveAsRawData`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in raw file format.

¬†

`void` **`SaveAsSmallHeader`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in Gatan small header file format.

¬†

`void` **`SaveAsText`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in text file format.

¬†

`void` **`SaveAsTIFF`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in TIFF file format. The DM Import/Export Plug-in must be installed in order for this function to work.

¬†

`void` **`SaveImage`**`( ImageReference, String` *`fileName`*` )`
Save the image to the fileName in it's current file format.

¬†

`void` **`SaveNotes`**`( ImageReference, String` *`tagPath`*`, String` *`fileName`*` )`
Saves the image notes indicated by tagPath into the given fileName. The tagPath must refer to a group.

¬†

`void` **`SavePersistentNotes`**`( String` *`tagPath`*`, String` *`fileName`*` )`
Saves the persistent notes indicated by tagPath into the given fileName. The tagPath must refer to a group.

¬†

`void` **`ScrapClear`**`( ImageReference )`
Clear the pasted image (if there is one) from the image.

¬†

`void` **`ScrapCopy`**`( ImageReference )`
Copy the selected portion of the image to the scrap.

¬†

`void` **`ScrapGetLocation`**`( ImageReference, NumberVariable` *`top`*`, NumberVariable` *`left`*` )`
Store the pasted image location (in image coordinates) into the top and left variables.

¬†

`void` **`ScrapGetSize`**`( ImageReference, NumberVariable` *`width`*`, NumberVariable` *`height`*` )`
Store the pasted image size (in image coordinates) into the width and height variables.

¬†

`void` **`ScrapMerge`**`( ImageReference )`
Merge the pasted image (if there is one) with the image.

¬†

`void` **`ScrapPaste`**`( ImageReference )`
Paste the scrap into the image.

¬†

`void` **`ScrapPasteNew`**`( )`
Paste the scrap into a new image.

¬†

`void` **`ScrapSetLocation`**`( ImageReference, Number` *`top`*`, Number` *`left`* `)`

Set the location of the pasted image (if there is one) to [top,left].

¬†

`void` **`ScreenGetBounds`**`( Number` *`index`*`, NumberVariable` *`t`*`, NumberVariable` *`l`*`, NumberVariable` *`b`*`, NumberVariable` *`r`* `)`

Gets the bounds of the 'index'th screen.

¬†

`void` **`ScreenGetWorkArea`**`( Number` *`index`*`, NumberVariable` *`t`*`, NumberVariable` *`l`*`, NumberVariable` *`b`*`, NumberVariable` *`r`* `)`

Gets the bounds of the 'index'th screen.

¬†

`void` **`ScriptInterfaceGenerateStubs`**`( String` *`interface_name`*`, String` *`str_h`*`, String` *`str_cp`* `)`

Generates C++ stubs for the interface.

¬†

`Number` **`ScriptObjectGetClassToken`**`( ScriptObject` *`scriptObject`*`, String` *`class_name`* `)`

Gets the token in 'scriptObject' corresponding to the class 'class_name'.

¬†

`Number` **`ScriptObjectGetID`**`( ScriptObject` *`scriptObject`* `)`

Returns a unique ID for this object. The object can be recovered by using GetScriptObjectFromID function.

¬†

`Boolean` **`ScriptObjectIsValid`**`( ScriptObject` *`scriptObject`* `)`

Returns true if 'scriptObject' references a valid object.

¬†

`Function` **`ScriptObjectLookupMethod`**`( ScriptObject` *`scriptObject`*`, Function` *`meth_abs`*`, String` *`class_name`* `)`

Returns the method of this object corresponding to the abstract method 'meth_abs' and class 'class_name'.

¬†

`Function` **`ScriptObjectLookupMethod`**`( ScriptObject` *`scriptObject`*`, Function` *`meth_abs`* `)`

Returns the specific method of this object corresponding to the abstract method 'meth_abs'.

¬†

`ScriptObject` **`ScriptObjectNullify`**`( ! )`

¬†

`String` **`ScriptToUserCoordinates`**`( ImageReference, Number` *`x`*`, Number` *`y`* `)`

Return a string representing the script coordinates.

¬†

`void` **`ScriptWindowExecute`**`( DocumentWindow` *`window`* `)`

Executes the script in the script window.

¬†

`void` **`SelectAnnotation`**`( ImageReference, Number` *`annotationID`* `)`

Select the annotation indicated by the annotationID within the image.

¬†

`void` **`SelectImage`**`( ImageReference )`

Bring the image's image document window to the front.

¬†

`void` **`SetAnnotationBackground`**`( ImageReference, Number` *`annotationID`*`, Number` *`background`* `)`

Set the background mode of the annotation indicated by the annotationID within the image. Possible values for background are 1=black-on-white, 2=black, 3=white-on-black, 4=white.

¬†

`void` **`SetAnnotationComplexNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, ComplexNumber` *`noteValue`* `)`

Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.

¬†

`void` **`SetAnnotationFace`**`( ImageReference, Number` *`annotationID`*`, Number` *`face`* `)`

Set the type face of the annotation indicated by the annotationID within the image. Values for the face can be found in InsideMacintosh Vol I.

¬†

`void` **`SetAnnotationFont`**`( ImageReference, Number` *`annotationID`*`, String` *`fontName`* `)`

Set the font of the annotation indicated by the annotationID within the image to fontname.

¬†

`void` **`SetAnnotationJustification`**`( ImageReference, Number` *`annotationID`*`, Number` *`justification`* `)`

Set the justification of the text annotation indicated by the annotationID within the image. Possible values for justification are -1=left, 0=center, 1=right.

¬†

`void` **`SetAnnotationNumberNote`**`( ImageReference, Number` *`annotationID`*`, String` *`noteLabel`*`, Number`

noteValue)
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetAnnotationRect**(ImageReference, Number *annotationID*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Set the bounds of the annotation indicated by the annotationID within the image to [top, left, bottom, right].
¬†
void **SetAnnotationRectNote**(ImageReference, Number *annotationID*, String *noteLabel*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the value of the annotation note with label noteLabel to the rectangle formed from the top, left, bottom, and right parameters. If the note does not exist, it is created.
¬†
void **SetAnnotationRGBNumberNote**(ImageReference, Number *annotationID*, String *noteLabel*, RGBNumber *noteValue*)
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetAnnotationSize**(ImageReference, Number *annotationID*, Number *size*)
Set the size of text of the annotation indicated by the annotationID within the image.
¬†
void **SetAnnotationStringNote**(ImageReference, Number *annotationID*, String *noteLabel*, String *noteValue*)
Sets the value of the annotation note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetApplicationFont**(Number *which_font*, String *face_name*, Number *attributes*, Number *size*, Number *text_encoding*)
Sets the selected appliction font to have the specified face name, attributes, size, and text encoding.
¬†
void **SetColorMode**(ImageReference, Number *mode*)
Set the color mode of the image. The possible values for mode are 1=greyscale, 3=rainbow, 4=temperature.
¬†
void **SetComplexMode**(ImageReference, Number *mode*)
Set the complex-to-real mode of the image. The possible value for mode are 1=real, 2=imaginary, 3=modulus, 4=log of modulus, 5=phase.
¬†
void **SetComplexNumberNote**(ImageReference, String *noteLabel*, ComplexNumber *noteValue*)
Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetContrastMode**(ImageReference, Number *mode*)
Set the constrast mode of the image. The possible value for mode are 1=linear, 2=equalized, 3=contoured, 4=custom.
¬†
void **SetCustomCLUT**(ImageReference, RGBImage *clutImage*)
Set the CLUT for the image to clutImage. clutImage must be a 256x1 RGB image.
¬†
void **SetDaemonComponent**(ImageReference, Number)
Set the daemon component for the image. See the SDK documentation for more information.
¬†
void **SetDisplayType**(ImageReference, Number *displayType*)
Set the display type of the image. Possible displayType values are -1=best, 1=raster image, 2=surface plot, 3=RGB, 4=line plot, 5=spreadsheet.
¬†
void **SetDoCreateCustomIcon**(ImageReference *image*, Boolean *doCreateCustomIcon*)
Open the image with the filename. Returns the opened image.
¬†
void **SetDoSavePreview**(ImageReference *image*, Boolean *doSavePreview*)
Sets whether to save a preview along with the saved file.
¬†
void **SetEstimatedMinMax**(ImageReference, Number *min*, Number *max*)
Set the estimated minimum and maximum of the image to min, max. This can be used if survey is turned off.
¬†
void **SetImage**(ImageReference *src*, RealImageVariable *dst*)
Assigns src to dst by reference.
¬†
void **SetImage**(ImageReference, ComplexImageVariable)
¬†
void **SetImagePositionWithinWindow**(ImageReference, Number *x*, Number *y*)
Set top-left position of the image to [x,y] within the image document.

¬†
void **SetInversionMode**(ImageReference, Boolean *inverted*)
Set the contrast of the image to be inverted or not inverted.
¬†
void **SetKeywordNote**(ImageReference, String *keyword*)
Adds the keyword to the image keyword list if it does not already exist.
¬†
void **SetLimits**(ImageReference, Number *low*, Number *high*)
Set the lowest and highest displayed pixel values for the image. Everything below low will be the 'black' color and every above high will be the 'white' color. The black and white colors may not actually be black and white if the color table is not greyscale.
¬†
void **SetMinContrast**(ImageReference, Number *minContrast*)
Set the minimum amount of contrast for the image to minContrast.
¬†
void **SetName**(ImageReference, String *name*)
Sets the name of the image's image document to name.
¬†
void **SetNoteState**(ImageReference, String *noteLabel*, Boolean *hidden*)
Sets whether the given image note is in the copy-to-image list. The hidden parameter is unused.
¬†
void **SetNumberNote**(ImageReference, String *noteLabel*, Number *noteValue*)
Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetOrigin**(ImageReference, Number *x*, Number *y*)
Set the origin of the image coordinates to [x,y] which are in pixel units.
¬†
void **SetPersistentComplexNumberNote**(String *noteLabel*, ComplexNumber *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPersistentKeywordNote**(String *keyword*)
Adds the keyword to the persistent keyword list if it does not already exist.
¬†
void **SetPersistentLongNote**(String *noteLabel*, Number *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPersistentNoteState**(String *noteLabel*, Boolean *copyToImage*, Boolean *hidden*)
Sets whether the given persistent note is in the copy-to-image list. The hidden parameter is unused.
¬†
void **SetPersistentNumberNote**(String *noteLabel*, Number *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPersistentPointNote**(String *noteLabel*, Number *x*, Number *y*)
Sets the value of the persistent note with label noteLabel to the point formed from the x and y parameters. If the note does not exist, it is created.
¬†
void **SetPersistentRectNote**(String *noteLabel*, Number *top*, Number *left*, Number *bottom*, Number *right*)
Sets the value of the persistent note with label noteLabel to the rectangle formed from the top, left, bottom, and right parameters. If the note does not exist, it is created.
¬†
void **SetPersistentRGBNumberNote**(String *noteLabel*, RGBNumber *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPersistentStringNote**(String *noteLabel*, String *noteValue*)
Sets the value of the persistent note with label noteLabel to noteValue. If the note does not exist, it is created.
¬†
void **SetPixel**(RGBImage, Number *x*, Number *y*, RGBNumber *value*)
Sets the pixel in the image at [x,y] to value.
¬†
void **SetPixel**(ComplexImage, Number *x*, Number *y*, ComplexNumber *value*)
Sets the pixel in the image at [x,y] to value. Does not work on packed complex images.
¬†
void **SetPixel**(RealImage, Number *x*, Number *y*, Number *value*)
Sets the pixel in the image at [x,y] to value.

¬†

void **SetPointNote**(ImageReference, String *noteLabel*, Number *x*, Number *y*)

Sets the value of the image note with label noteLabel to the point formed from the x and y parameters. If the note does not exist, it is created.

¬†

void **SetRawStreamPos**(Number *rawStream*, Number *mode*, Number *offset*)

Set the current position in rawStream to offset using the mode.

¬†

void **SetRectNote**(ImageReference, String *noteLabel*, Number *top*, Number *left*, Number *bottom*, Number *right*)

Sets the value of the image note with label noteLabel to the rectangle formed from the top, left, bottom, and right parameters. If the note does not exist, it is created.

¬†

void **SetRGBNumberNote**(ImageReference, String *noteLabel*, RGBNumber *noteValue*)

Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.

¬†

void **SetScale**(ImageReference, Number *x*, Number *y*)

Set the scale of the image to [x,y].

¬†

void **SetSelection**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)

Sets the selection of the image to [top,left,bottom,right].

¬†

void **SetStarveRegisters**(Boolean)

¬†

void **SetStringNote**(ImageReference, String *noteLabel*, String *noteValue*)

Sets the value of the image note with label noteLabel to noteValue. If the note does not exist, it is created.

¬†

void **SetSurvey**(ImageReference, Boolean *surveyOnOff*)

Turn surveying on or off for the image.

¬†

void **SetSurveyTechnique**(ImageReference, Number *mode*)

Set the survey technique for the image to mode. Mode can be 0=cross-wire, 1=whole image, 2=sparse.

¬†

void **SetUnitString**(ImageReference, String *unitString*)

Set the units of the image to unitString.

¬†

void **SetWindowBounds**(ImageReference, Number *top*, Number *left*, Number *bottom*, Number *right*)

Set the image's image document window bounds to (left,top),(right,bottom).

¬†

void **SetWindowPosition**(ImageReference, Number *xPos*, Number *yPos*)

Set the image's image document window position to [xPos, yPos]. Only valid for images that are already shown in a window.

¬†

void **SetWindowSize**(ImageReference, Number *width*, Number *height*)

Set the image's image document window size to [width, height].

¬†

void **SetZoom**(ImageReference, Number *zoom*)

Set the zoom of the image display.

¬†

RealNumberExpression **sgn**(RealNumberExpression)

Returns 1 if the number is equal to or greater than 0 otherwise returns -1.

¬†

Number **sgn**(Number *v1*)

¬†

void **ShiftCenter**(ImageReference *image*)

Shifts each dimension of an image by half. For two dimensional images it will swap quadrants.

¬†

Boolean **ShiftDown**( )

Returns 1 if the shift key is down and 0 otherwise.

¬†

void **ShowAlert**(String *prompt*, Number *alertStyle*)

Puts up an alert with the given prompt and style.

¬†

void **ShowImage**(ImageReference)

Display the image's image document in a window if it is not displayed already.

¬†

ComplexNumberExpression **sin**(ComplexNumberExpression)
Return the sine of the number.
¬†
RealNumberExpression **sin**(RealNumberExpression)
Return the sine of the number.
¬†
Number **sin**(Number *v1*)
¬†
ComplexNumberExpression **sinh**(ComplexNumberExpression)
Return the hyperbolic sine of the number.
¬†
Number **sinh**(Number *v1*)
¬†
RealNumberExpression **sinh**(RealNumberExpression)
Return the hyberbolic sine of the number.
¬†
void **Sleep**(Number *seconds*)
Puts the current thread to sleep for the given number of seconds (resolution may vary by platform).
¬†
BasicImage **slice1**(BasicImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*)
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
RGBImage **slice1**(RGBImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*)
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
RealImage **slice1**(RealImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*)
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
ComplexImage **slice1**(ComplexImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*)
Returns the subslice [(x,y,z),(d0,l0,s0)].
¬†
BasicImage **slice2**(BasicImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
RealImage **slice2**(RealImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
ComplexImage **slice2**(ComplexImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
RGBImage **slice2**(RGBImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1)].
¬†
RGBImage **slice3**(RGBImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
BasicImage **slice3**(BasicImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
ComplexImage **slice3**(ComplexImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
RealImage **slice3**(RealImage, Number *x*, Number *y*, Number *z*, Number *d0*, Number *l0*, Number *s0*, Number *d1*, Number *l1*, Number *s1*, Number *d2*, Number *l2*, Number *s2*)
Returns the subslice [(x,y,z),(d0,l0,s0),(d1,l1,s1),(d2,l2,s2)].
¬†
Boolean **SpaceDown**( )
Returns 1 if the space key is down and 0 otherwise.
¬†

RealNumberExpression **SphericalBesselJ**(Number *order*, Number)
Return the spherical Bessel J function of the given order for a number.
¬†

RealNumberExpression **SphericalBesselY**(Number, Number)
Return the spherical Bessel Y function of the given order for a number.
¬†

ComplexNumberExpression **sqrt**(ComplexNumberExpression)
Return the square root of the number.
¬†

Number **sqrt**(Number *v1*)
¬†

RealNumberExpression **sqrt**(RealNumberExpression)
Return the square root of the number.
¬†

void **StopAcquisitionDaemon**(ImageReference)
Stop all acquisition daemons associated with the image.
¬†

void **StopAllAcquisitionDaemons**( )
Stop all acquisition daemons with any image within the application.
¬†

Number **StreamGetPos**(ScriptObject *stream*)
Returns the current position within the file.
¬†

Number **StreamGetSize**(ScriptObject *stream*)
Returns the size of the file.
¬†

String **StreamReadAsText**(ScriptObject *stream*, Number *encoding*, Number *count*)
Reads count bytes from a file, returning them as a string assuming they have the specified encoding.
¬†

Boolean **StreamReadTextLine**(ScriptObject *stream*, Number *encoding*, String *str_out*)
Read a line of text from the stream, storing it into the string variable, assuming the text has the specified encoding. Return 1 if successful and 0 otherwise.
¬†

void **StreamSetPos**(ScriptObject *stream*, Number *base*, Number *offset*)
Sets the current position within the file as an offset from 'base', where base == 0 denotes beginning of file, 1 denotes current position, and 2 denotes end of file.
¬†

void **StreamSetSize**(ScriptObject *stream*, Number *size*)
Sets the size of the file.
¬†

void **StreamWriteAsText**(ScriptObject *stream*, Number *encoding*, String *data*)
Write the string to the file with the specified encoding.
¬†

String **StringAppend**(String *s1*, String *s2*)
Appends string s2 to s1, converting its encoding to that of s1 if necessary.
¬†

String **StringAppend**(String *s1*, Number *ch*, Number *encoding_id*)
Appends character ch to s1, converting its encoding to that of s1 if necessary.
¬†

Number **StringCompare**(String *s1*, String *s2*)
Compares strings 's1' and 's2', returning -1,0, or 1 if s1 is less, equal, or greater than s2.
¬†

String **StringConvertToEncoding**(String *s1*, Number *encoding_id*)
Converts 's1' to the encoding specified by 'encoding_id'.
¬†

Boolean **StringIsValid**(String *str*)
Returns true if 'str' is a valid object.
¬†

String **StringNullify**(!)
Assigns NULL to dst string.
¬†

ComplexNumber **sum**(ComplexImageExpression)
Return the sum of all pixel values of the image expression.
¬†

Number **sum**(RealImageExpression)
Return the sum of all pixel values of the image expression.
¬†
void **SurfacePlotImageDisplayGetCubeAxes**(SurfacePlotImageDisplay *spid*, NumberVariable *x_axis_x*, NumberVariable *x_axis_y*, NumberVariable *y_axis_x*, NumberVariable *y_axis_y*, NumberVariable *z_axis*)
Gets the points describing the surface plot cube.
¬†
void **SurfacePlotImageDisplayGetCubePoint**(SurfacePlotImageDisplay *spid*, Number *which_point*, NumberVariable *x*, NumberVariable *y*)
Gets the child coordinates of the cube point indicated by 'which_point'
¬†
Boolean **SurfacePlotImageDisplayIsShadingOn**(SurfacePlotImageDisplay *spid*)
Determines whether shading is on or off.
¬†
SurfacePlotImageDisplay **SurfacePlotImageDisplayNullify**(!)
¬†
void **SurfacePlotImageDisplaySetCubeAxes**(SurfacePlotImageDisplay *spid*, Number *x_axis_x*, Number *x_axis_y*, Number *y_axis_x*, Number *y_axis_y*, Number *z_axis*)
Sets the points describing the surface plot cube.
¬†
void **SurfacePlotImageDisplaySetShadingOn**(SurfacePlotImageDisplay *spid*, Boolean *on*)
Sets whether shading is on or off.
¬†
RealImage **SVDecomposition**(RealImage *a*, RealImage *b*)
Return the image resulting from a SV decomposition on images a,b.
¬†
RealImage **SVDFit**(RealImage *a*, RealImage *b*, Number *tolerance*)
Return the image resulting from a SVD fit on images a,b with the given tolerance.
¬†
void **swap**(RealImageExpression *a*, RealImageExpression *b*)
Swap images a and b, pixel by pixel.
¬†
void **swap**(ComplexImageExpression *a*, ComplexImageExpression *b*)
Swap images a and b, pixel by pixel.
¬†
void **swap**(RGBImageExpression *a*, RGBImageExpression *b*)
Swap images a and b, pixel by pixel.
¬†
void **SwapByteOrder**(ImageReference)
Swaps the byte order for each long word in the image. ABCD become DCBA.
¬†
void **SwapWordOrder**(ImageReference)
Swaps the byte order for each short word in the image. ABCD become BADC.
¬†
TagGroup **TagGroupAddLabeledTagGroup**(TagGroup *tagGroup*, String *label*, TagGroup *newGroup*)
Adds 'newGroup' to 'tagGroup' at the label 'label'.
¬†
TagGroup **TagGroupAddTagGroupAfter**(TagGroup *tagList*, Number *ref_index*, TagGroup *newGroup*)
Adds 'newGroup' to 'tagList' after index 'ref_index'.
¬†
TagGroup **TagGroupAddTagGroupAtBeginning**(TagGroup *tagList*, TagGroup *newGroup*)
Adds 'newGroup' to the beginning of 'tagList'.
¬†
TagGroup **TagGroupAddTagGroupAtEnd**(TagGroup *tagList*, TagGroup *newGroup*)
Adds 'newGroup' to the end of 'tagList'.
¬†
TagGroup **TagGroupAddTagGroupBefore**(TagGroup *tagList*, Number *ref_index*, TagGroup *newGroup*)
Adds 'newGroup' to 'tagList' before index 'ref_index'.
¬†
TagGroup **TagGroupClone**(TagGroup *tagGroup*)
Returns an identical copy of 'tagGroup' and its sub-tags.
¬†
Number **TagGroupCopyTag**(TagGroup *tagGroup*, TagGroup *srcGroup*, Number *srcIndex*)
Copies the 'srcIndex'th tag in 'srcGroup' to 'tagGroup'.
¬†

void **TagGroupCopyTagsFrom**(TagGroup *tagGroup*, TagGroup *srcGroup*)
Copies tags in 'srcGroup' to 'tagGroup'.
¬†
void **TagGroupCopyTagToIndex**(TagGroup *tagGroup*, Number *dstIndex*, TagGroup *srcGroup*, Number *srcIndex*)
Copies data in the 'srcIndex'th tag in 'srcGroup' to the 'dstIndex'th tag in 'tagGroup'.
¬†
Number **TagGroupCountTags**(TagGroup *tagGroup*)
Returns the number of sub-tags in this tag group.
¬†
TagGroup **TagGroupCreateGroupTagAfter**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group after 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateGroupTagAtBeginning**(TagGroup *tagList*)
Creates a new tag group at the beginning of 'tagList'.
¬†
TagGroup **TagGroupCreateGroupTagAtEnd**(TagGroup *tagList*)
Creates a new tag group at the end of 'tagList'.
¬†
TagGroup **TagGroupCreateGroupTagBefore**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group before 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateListTagAfter**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group after 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateListTagAtBeginning**(TagGroup *tagList*)
Creates a new tag group at the beginning of 'tagList'.
¬†
TagGroup **TagGroupCreateListTagAtEnd**(TagGroup *tagList*)
Creates a new tag group at the end of 'tagList'.
¬†
TagGroup **TagGroupCreateListTagBefore**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag group before 'ref_index' in 'tagList'.
¬†
TagGroup **TagGroupCreateNewLabeledGroup**(TagGroup *tagGroup*, String *label*)
Adds a new tag group at label 'label' and returns the new group.
¬†
TagGroup **TagGroupCreateNewLabeledList**(TagGroup *tagGroup*, String *label*)
Adds a new tag list at label 'label' and returns the new group.
¬†
Number **TagGroupCreateNewLabeledTag**(TagGroup *tagGroup*, String *label*)
Creates a new labeled tag and returns its index.
¬†
Number **TagGroupCreateNewTagAfter**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag after 'ref_index' in 'tagList'.
¬†
Number **TagGroupCreateNewTagAtBeginning**(TagGroup *tagList*)
Creates a new tag at the beginning of 'tagList'.
¬†
Number **TagGroupCreateNewTagAtEnd**(TagGroup *tagList*)
Creates a new tag at the end of 'tagList'.
¬†
Number **TagGroupCreateNewTagBefore**(TagGroup *tagList*, Number *ref_index*)
Creates a new tag before 'ref_index' in 'tagList'.
¬†
void **TagGroupDeleteAllTags**(TagGroup *tagGroup*)
Deletes all the tags in 'tagGroup'.
¬†
void **TagGroupDeleteTagWithIndex**(TagGroup *taGroup*, Number *index*)
Deletes the tag at index 'index'.
¬†
void **TagGroupDeleteTagWithLabel**(TagGroup *tagGroup*, String *tagPath*)
Deletes the tag labelled by the path 'tagPath'.
¬†
Boolean **TagGroupDoesTagExist**(TagGroup *tagGroup*, String *tagPath*)

Finds the tag group and index corresponding to the tag referenced by 'tagPath' in 'tagGroup'.
¬†

`void` **`TagGroupExecuteScriptGroup`**`( TagGroup` *`tagGroup`*`, String` *`form`*` )`
Execute a group of script functions in 'tagGroup'. The actual scripts executed will be formed by sprintf'ing into the 'form' parameter. The form parameter should contain exactly one '%s' into which the function name will be inserted.
¬†

`Boolean` **`TagGroupGetIndexedTagAsArray`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, ImageReference` *`image`*` )`
Gets the data at 'index' in 'tagGroup' as an array of data in 'image'.
¬†

`Boolean` **`TagGroupGetIndexedTagAsBoolean`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`val`*` )`
Gets the data at 'index' in 'tagGroup' as a boolean.
¬†

`Boolean` **`TagGroupGetIndexedTagAsDouble`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a double.
¬†

`Boolean` **`TagGroupGetIndexedTagAsDoubleComplex`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, ComplexNumberVariable` *`c`*` )`
Gets the data at 'index' in 'tagGroup' as a double complex.
¬†

`Boolean` **`TagGroupGetIndexedTagAsEightBitColor`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, RGBNumberVariable` *`c`*` )`
Gets the data at 'index' in 'tagGroup' as an eight bit color.
¬†

`Boolean` **`TagGroupGetIndexedTagAsFloat`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a float.
¬†

`Boolean` **`TagGroupGetIndexedTagAsFloatComplex`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, ComplexNumberVariable` *`c`*` )`
Gets the data at 'index' in 'tagGroup' as a float complex.
¬†

`Boolean` **`TagGroupGetIndexedTagAsFloatPoint`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Gets the data at 'index' in 'tagGroup' as a float point.
¬†

`Boolean` **`TagGroupGetIndexedTagAsFloatRect`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`t`*`, NumberVariable` *`l`*`, NumberVariable` *`b`*`, NumberVariable` *`r`*` )`
Gets the data at 'index' in 'tagGroup' as a float rect.
¬†

`Boolean` **`TagGroupGetIndexedTagAsLong`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a long.
¬†

`Boolean` **`TagGroupGetIndexedTagAsLongPoint`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Gets the data at 'index' in 'tagGroup' as a long point.
¬†

`Boolean` **`TagGroupGetIndexedTagAsLongRect`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`t`*`, NumberVariable` *`l`*`, NumberVariable` *`b`*`, NumberVariable` *`r`*` )`
Gets the data at 'index' in 'tagGroup' as a long rect.
¬†

`Boolean` **`TagGroupGetIndexedTagAsNumber`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a real number.
¬†

`Boolean` **`TagGroupGetIndexedTagAsNumber`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, ComplexNumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a complex number.
¬†

`Boolean` **`TagGroupGetIndexedTagAsNumber`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, RGBNumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a RGB number.
¬†

`Boolean` **`TagGroupGetIndexedTagAsShort`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`number`*` )`
Gets the data at 'index' in 'tagGroup' as a short.
¬†

`Boolean` **`TagGroupGetIndexedTagAsShortPoint`**`( TagGroup` *`tagGroup`*`, Number` *`index`*`, NumberVariable` *`x`*`, NumberVariable` *`y`*` )`
Gets the data at 'index' in 'tagGroup' as a short point.
¬†

Boolean **TagGroupGetIndexedTagAsShortRect**(TagGroup *tagGroup*, Number *index*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'index' in 'tagGroup' as a short rect.
¬†

Boolean **TagGroupGetIndexedTagAsString**(TagGroup *tagGroup*, Number *index*, String *str*)
Gets the data at 'index' in 'tagGroup' as a string.
¬†

Boolean **TagGroupGetIndexedTagAsTagGroup**(TagGroup *tagGroup*, Number *index*, TagGroup *subGroup*)
Gets the data at 'index' in 'TagGroup' as a group.
¬†

Boolean **TagGroupGetIndexedTagAsText**(TagGroup *tagGroup*, Number *index*, String *str*)
Gets the data at 'index' in 'tagGroup' as a string.
¬†

Boolean **TagGroupGetIndexedTagAsUInt16**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a 16-bit unsigned integer.
¬†

Boolean **TagGroupGetIndexedTagAsUInt32**(TagGroup *tagGroup*, Number *index*, NumberVariable *number*)
Gets the data at 'index' in 'tagGroup' as a 32-bit unsigned integer.
¬†

TagGroup **TagGroupGetOrCreateTagGroup**(TagGroup *tagGroup*, String *tagPath*)
Gets the tag group named by 'tagPath', or creates a new such group and all necessary intermediate groups.
¬†

TagGroup **TagGroupGetOrCreateTagList**(TagGroup *tagGroup*, String *tagPath*)
Gets the tag list named by 'tagPath', or creates a new such list and all necessary intermediate groups.
¬†

Number **TagGroupGetSeeds**(TagGroup *tagGroup*)
Gets a set of seeds that describe the tag group.
¬†

Boolean **TagGroupGetTagAsArray**(TagGroup *tagGroup*, String *tagPath*, ImageReference *image*)
Gets the data at 'tagPath' in 'tagGroup' as an array of data in 'image'.
¬†

Boolean **TagGroupGetTagAsBoolean**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *val*)
Gets the data at 'tagPath' in 'tagGroup' as a boolean.
¬†

Boolean **TagGroupGetTagAsDouble**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a double.
¬†

Boolean **TagGroupGetTagAsDoubleComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumberVariable *c*)
Gets the data at 'tagPath' in 'tagGroup' as a double complex.
¬†

Boolean **TagGroupGetTagAsEightBitColor**(TagGroup *tagGroup*, String *tagPath*, RGBNumberVariable *c*)
Gets the data at 'tagPath' in 'tagGroup' as an eight bit color.
¬†

Boolean **TagGroupGetTagAsFloat**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a float.
¬†

Boolean **TagGroupGetTagAsFloatComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumberVariable *c*)
Gets the data at 'tagPath' in 'tagGroup' as a float complex.
¬†

Boolean **TagGroupGetTagAsFloatPoint**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'tagPath' in 'tagGroup' as a short point.
¬†

Boolean **TagGroupGetTagAsFloatRect**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'tagPath' in 'tagGroup' as a short rect.
¬†

Boolean **TagGroupGetTagAsLong**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a long.
¬†

Boolean **TagGroupGetTagAsLongPoint**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'tagPath' in 'tagGroup' as a long point.
¬†

Boolean **TagGroupGetTagAsLongRect**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *t*,

NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'tagPath' in 'tagGroup' as a long rect.
¬†
Boolean **TagGroupGetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, ComplexNumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a complex number.
¬†
Boolean **TagGroupGetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a real number.
¬†
Boolean **TagGroupGetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, RGBNumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a rgb number.
¬†
Boolean **TagGroupGetTagAsRGBUInt16**(TagGroup *tagGroup*, Number *index*, NumberVariable *r*, NumberVariable *g*, NumberVariable *b*)
Gets the data at 'index' in 'tagGroup' as a 16-bit rgb value.
¬†
Boolean **TagGroupGetTagAsRGBUInt16**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *r*, NumberVariable *g*, NumberVariable *b*)
Gets the data at 'tagPath' in 'tagGroup' as a 16-bit rgb value.
¬†
Boolean **TagGroupGetTagAsShort**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a short.
¬†
Boolean **TagGroupGetTagAsShortPoint**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *x*, NumberVariable *y*)
Gets the data at 'tagPath' in 'tagGroup' as a short point.
¬†
Boolean **TagGroupGetTagAsShortRect**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *t*, NumberVariable *l*, NumberVariable *b*, NumberVariable *r*)
Gets the data at 'tagPath' in 'tagGroup' as a short rect.
¬†
Boolean **TagGroupGetTagAsString**(TagGroup *tagGroup*, String *tagPath*, String *str*)
Gets the data at 'tagPath' in 'tagGroup' as a string.
¬†
Boolean **TagGroupGetTagAsTagGroup**(TagGroup *tagGroup*, String *tagPath*, TagGroup *subGroup*)
Gets the data at 'tagPath' in 'TagGroup' as a group.
¬†
Boolean **TagGroupGetTagAsText**(TagGroup *tagGroup*, String *tagPath*, String *str*)
Gets the data at 'tagPath' in 'tagGroup' as a string.
¬†
Boolean **TagGroupGetTagAsUInt16**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a 16-bit unsigned integer.
¬†
Boolean **TagGroupGetTagAsUInt32**(TagGroup *tagGroup*, String *tagPath*, NumberVariable *number*)
Gets the data at 'tagPath' in 'tagGroup' as a 32-bit unsigned integer.
¬†
String **TagGroupGetTagLabel**(TagGroup *tagGroup*, Number *index*)
Gets the label of the 'index'th tag in the tag group.
¬†
Number **TagGroupGetTagSize**(TagGroup *tagGroup*, Number *index*)
Gets the size of the tag.
¬†
Number **TagGroupGetTagType**(TagGroup *tagGroup*, Number *index*, Number *type_index*)
Returns the 'type_index'th element of the tag's type.
¬†
Number **TagGroupGetTagTypeLength**(TagGroup *tagGroup*, Number *index*)
Returns number of elements in the tag's type.
¬†
Boolean **TagGroupHasChangedSince**(TagGroup *tagGroup*, Number *seeds*)
Returns true if the tag group has changed since 'seeds' was constructed.
¬†
void **TagGroupInsertTagAsArray**(TagGroup *tagGroup*, Number *ref_index*, ImageReference *image*)
Inserts new data before 'ref_index' in 'tagGroup' as an array of data in 'image'.
¬†
void **TagGroupInsertTagAsBoolean**(TagGroup *tagGroup*, Number *ref_index*, Boolean *val*)

Inserts new data before 'ref_index' in 'tagGroup' as a boolean.
¬†
void **TagGroupInsertTagAsDouble**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a double.
¬†
void **TagGroupInsertTagAsDoubleComplex**(TagGroup *tagGroup*, Number *ref_index*, ComplexNumber *c*)
Inserts new data before 'ref_index' in 'tagGroup' as a double complex.
¬†
void **TagGroupInsertTagAsEightBitColor**(TagGroup *tagGroup*, Number *ref_index*, RGBNumber *c*)
Inserts new data before 'ref_index' in 'tagGroup' as an eight bit color.
¬†
void **TagGroupInsertTagAsFloat**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a float.
¬†
void **TagGroupInsertTagAsFloatComplex**(TagGroup *tagGroup*, Number *ref_index*, ComplexNumber *c*)
Inserts new data before 'ref_index' in 'tagGroup' as a float complex.
¬†
void **TagGroupInsertTagAsFloatPoint**(TagGroup *tagGroup*, Number *ref_index*, Number *x*, Number *y*)
Inserts new data before 'ref_index' in 'tagGroup' as a float point.
¬†
void **TagGroupInsertTagAsFloatRect**(TagGroup *tagGroup*, Number *ref_index*, Number *t*, Number *l*, Number *b*, Number *r*)
Inserts new data before 'ref_index' in 'tagGroup' as a float rect.
¬†
void **TagGroupInsertTagAsLong**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a long.
¬†
void **TagGroupInsertTagAsLongPoint**(TagGroup *tagGroup*, Number *ref_index*, Number *x*, Number *y*)
Inserts new data before 'ref_index' in 'tagGroup' as a long point.
¬†
void **TagGroupInsertTagAsLongRect**(TagGroup *tagGroup*, Number *ref_index*, Number *t*, Number *l*, Number *b*, Number *r*)
Inserts new data before 'ref_index' in 'tagGroup' as a long rect.
¬†
void **TagGroupInsertTagAsNumber**(TagGroup *tagGroup*, Number *ref_index*, RGBNumber *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a RGB number.
¬†
void **TagGroupInsertTagAsNumber**(TagGroup *tagGroup*, Number *ref_index*, ComplexNumber *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a complex number.
¬†
void **TagGroupInsertTagAsNumber**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a real number.
¬†
void **TagGroupInsertTagAsRGBUInt16**(TagGroup *tagGroup*, Number *ref_index*, Number *r*, Number *g*, Number *b*)
Inserts new data before 'ref_index' in 'tagGroup' as a 16-bit rgb value.
¬†
void **TagGroupInsertTagAsShort**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a short.
¬†
void **TagGroupInsertTagAsShortPoint**(TagGroup *tagGroup*, Number *ref_index*, Number *x*, Number *y*)
Inserts new data before 'ref_index' in 'tagGroup' as a short point.
¬†
void **TagGroupInsertTagAsShortRect**(TagGroup *tagGroup*, Number *ref_index*, Number *t*, Number *l*, Number *b*, Number *r*)
Inserts new data before 'ref_index' in 'tagGroup' as a short rect.
¬†
void **TagGroupInsertTagAsString**(TagGroup *tagGroup*, Number *ref_index*, String *s*)
Inserts new data before 'ref_index' in 'tagGroup' as a string.
¬†
void **TagGroupInsertTagAsTagGroup**(TagGroup *tagGroup*, Number *ref_index*, TagGroup *subGroup*)
Inserts new data before 'ref_index' in 'TagGroup' as a group.
¬†
void **TagGroupInsertTagAsText**(TagGroup *tagGroup*, Number *ref_index*, String *s*)
Inserts new data before 'ref_index' in 'tagGroup' as a string.

¬†
void **TagGroupInsertTagAsUInt16**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a 16-bit unsigned integer.
¬†
void **TagGroupInsertTagAsUInt32**(TagGroup *tagGroup*, Number *ref_index*, Number *number*)
Inserts new data before 'ref_index' in 'tagGroup' as a 32-bit unsigned integer.
¬†
Boolean **TagGroupIsList**(TagGroup *tagGroup*)
Returns true if the tag group is a list.
¬†
Boolean **TagGroupIsOpen**(TagGroup *tagGroup*)
Returns whether 'tagGroup' is open or not.
¬†
Boolean **TagGroupIsValid**(TagGroup *tagGroup*)
Returns true if 'tagGroup' references a valid object.
¬†
Boolean **TagGroupLoadFromFile**(TagGroup *tagGroup*, String *path*)
Loads the contents of the file specified by 'path' into the tag group.
¬†
Boolean **TagGroupLoadFromFileWithLabel**(TagGroup *tagGroup*, String *path*, String *label*)
Loads the contents of the file specified by 'path' into the tag group and returns the label, if any.
¬†
Boolean **TagGroupLoadFromStream**(TagGroup *tagGroup*, ScriptObject *stream*)
Loads the contents of the stream specified by 'stream' into the tag group.
¬†
Boolean **TagGroupLoadFromStreamWithLabel**(TagGroup *tagGroup*, ScriptObject *stream*, String *label*)
Loads the contents of the stream specified by 'stream' into the tag group and returns the label, if any.
¬†
void **TagGroupMarkAsChanged**(TagGroup *tagGroup*)
Marks 'tagGroup' as having beein modified.
¬†
TagGroup **TagGroupNullify**(!)
¬†
DocumentWindow **TagGroupOpenBrowserWindow**(TagGroup *tagGroup*, Boolean *isFileBased*)
Opens a browser window for the tag group.
¬†
Number **TagGroupParseAndCreateTagPath**(TagGroup *tagGroup*, String *tagPath*, TagGroup *parentGroup*, String *label*)
Finds the tag group and index corresponding to the tag referenced by 'tagPath' in 'tagGroup'.
¬†
Number **TagGroupParseTagPath**(TagGroup *tagGroup*, String *tagPath*, TagGroup *parentGroup*, String *label*)
Finds the tag group and index corresponding to the tag referenced by 'tagPath' in 'tagGroup'.
¬†
void **TagGroupReadIndexedTagDataFromStream**(TagGroup *tagGroup*, Number *index*, ScriptObject *stream*, Number *stream_endianness*)
Reads data from the stream into the indicated tag. The tag type determines how the data is read, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
void **TagGroupReadTagDataFromStream**(TagGroup *tagGroup*, String *tag_path*, ScriptObject *stream*, Number *stream_endianness*)
Reads data from the stream into the indicated tag. The tag type determines how the data is read, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
void **TagGroupReleaseSeeds**(TagGroup *tagGroup*, Number *seeds*)
Releases the seeds returned by 'TagGroupGetSeeds'.
¬†
void **TagGroupReplaceTagsWithCopy**(TagGroup *tagGroup*, TagGroup *srcGroup*)
Deletes all tags in 'tagGroup' and copies tags in 'srcGroup' to 'tagGroup'.
¬†
void **TagGroupSaveToFile**(TagGroup *tagGroup*, String *path*)
Saves the contents of the tag group to the file specified by 'path'.
¬†
void **TagGroupSaveToFileWithLabel**(TagGroup *tagGroup*, String *path*, String *label*)
Saves the contents of the tag group and the label 'label' to the file specified by 'path'.

¬†
void **TagGroupSaveToStream**(TagGroup *tagGroup*, ScriptObject *stream*)
Saves the contents of the tag group to the stream specified by 'stream'.
¬†
void **TagGroupSaveToStreamWithLabel**(TagGroup *tagGroup*, ScriptObject *stream*, String *label*)
Saves the contents of the tag group and the label 'label' to the stream specified by 'stream'.
¬†
void **TagGroupSetIndexedTagAsArray**(TagGroup *tagGroup*, Number *index*, ImageReference *image*)
Set the data at 'index' in 'tagGroup' as an array of data in 'image'.
¬†
void **TagGroupSetIndexedTagAsBoolean**(TagGroup *tagGroup*, Number *index*, Boolean *val*)
Sets the data at 'index' in 'tagGroup' as a boolean.
¬†
void **TagGroupSetIndexedTagAsDouble**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a double.
¬†
void **TagGroupSetIndexedTagAsDoubleComplex**(TagGroup *tagGroup*, Number *index*, ComplexNumber *c*)
Sets the data at 'index' in 'tagGroup' as a double complex.
¬†
void **TagGroupSetIndexedTagAsEightBitColor**(TagGroup *tagGroup*, Number *index*, RGBNumber *c*)
Sets the data at 'index' in 'tagGroup' as an eight bit color.
¬†
void **TagGroupSetIndexedTagAsFloat**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a float.
¬†
void **TagGroupSetIndexedTagAsFloatComplex**(TagGroup *tagGroup*, Number *index*, ComplexNumber *c*)
Sets the data at 'index' in 'tagGroup' as a float complex.
¬†
void **TagGroupSetIndexedTagAsFloatPoint**(TagGroup *tagGroup*, Number *index*, Number *x*, Number *y*)
Sets the data at 'index' in 'tagGroup' as a float point.
¬†
void **TagGroupSetIndexedTagAsFloatRect**(TagGroup *tagGroup*, Number *index*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'index' in 'tagGroup' as a float rect.
¬†
void **TagGroupSetIndexedTagAsLong**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a long.
¬†
void **TagGroupSetIndexedTagAsLongPoint**(TagGroup *tagGroup*, Number *index*, Number *x*, Number *y*)
Sets the data at 'index' in 'tagGroup' as a long point.
¬†
void **TagGroupSetIndexedTagAsLongRect**(TagGroup *tagGroup*, Number *index*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'index' in 'tagGroup' as a long rect.
¬†
void **TagGroupSetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, RGBNumber *number*)
Sets the data at 'index' in 'tagGroup' as a RGB number.
¬†
void **TagGroupSetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, ComplexNumber *number*)
Sets the data at 'index' in 'tagGroup' as a complex number.
¬†
void **TagGroupSetIndexedTagAsNumber**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a real number.
¬†
void **TagGroupSetIndexedTagAsRGBUInt16**(TagGroup *tagGroup*, Number *index*, Number *r*, Number *g*, Number *b*)
Sets the data at 'index' in 'tagGroup' as a 16-bit rgb value.
¬†
void **TagGroupSetIndexedTagAsShort**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a short.
¬†
void **TagGroupSetIndexedTagAsShortPoint**(TagGroup *tagGroup*, Number *index*, Number *x*, Number *y*)
Sets the data at 'index' in 'tagGroup' as a short point.
¬†
void **TagGroupSetIndexedTagAsShortRect**(TagGroup *tagGroup*, Number *index*, Number *t*, Number *l*, Number

b, Number *r*)
Sets the data at 'index' in 'tagGroup' as a short rect.
¬†
void **TagGroupSetIndexedTagAsString**(TagGroup *tagGroup*, Number *index*, String *s*)
Sets the data at 'index' in 'tagGroup' as a string.
¬†
void **TagGroupSetIndexedTagAsTagGroup**(TagGroup *tagGroup*, Number *index*, TagGroup *subGroup*)
Sets the data at 'index' in 'TagGroup' as a group.
¬†
void **TagGroupSetIndexedTagAsText**(TagGroup *tagGroup*, Number *index*, String *s*)
Sets the data at 'index' in 'tagGroup' as a string.
¬†
void **TagGroupSetIndexedTagAsUInt16**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a 16-bit unsigned integer.
¬†
void **TagGroupSetIndexedTagAsUInt32**(TagGroup *tagGroup*, Number *index*, Number *number*)
Sets the data at 'index' in 'tagGroup' as a 32-bit unsigned integer.
¬†
void **TagGroupSetIsOpen**(TagGroup *tagGroup*, Boolean *is_open*)
Sets whether 'tagGroup' is open or not.
¬†
void **TagGroupSetTagAsArray**(TagGroup *tagGroup*, String *tagPath*, ImageReference *image*)
Set the data at 'tagPath' in 'tagGroup' as an array of data in 'image'.
¬†
void **TagGroupSetTagAsBoolean**(TagGroup *tagGroup*, String *tagPath*, Boolean *val*)
Sets the data at 'tagPath' in 'tagGroup' as a boolean.
¬†
void **TagGroupSetTagAsDouble**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a double.
¬†
void **TagGroupSetTagAsDoubleComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumber *c*)
Sets the data at 'tagPath' in 'tagGroup' as a double complex.
¬†
void **TagGroupSetTagAsEightBitColor**(TagGroup *tagGroup*, String *tagPath*, RGBNumber *c*)
Sets the data at 'tagPath' in 'tagGroup' as an eight bit color.
¬†
void **TagGroupSetTagAsFloat**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a float.
¬†
void **TagGroupSetTagAsFloatComplex**(TagGroup *tagGroup*, String *tagPath*, ComplexNumber *c*)
Sets the data at 'tagPath' in 'tagGroup' as a float complex.
¬†
void **TagGroupSetTagAsFloatPoint**(TagGroup *tagGroup*, String *tagPath*, Number *x*, Number *y*)
Sets the data at 'tagPath' in 'tagGroup' as a float point.
¬†
void **TagGroupSetTagAsFloatRect**(TagGroup *tagGroup*, String *tagPath*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'tagPath' in 'tagGroup' as a float rect.
¬†
void **TagGroupSetTagAsLong**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a long.
¬†
void **TagGroupSetTagAsLongPoint**(TagGroup *tagGroup*, String *tagPath*, Number *x*, Number *y*)
Sets the data at 'tagPath' in 'tagGroup' as a long point.
¬†
void **TagGroupSetTagAsLongRect**(TagGroup *tagGroup*, String *tagPath*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'tagPath' in 'tagGroup' as a long rect.
¬†
void **TagGroupSetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, RGBNumber *number*)
Sets the data at 'tagPath' in 'tagGroup' as a RGB number.
¬†
void **TagGroupSetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, ComplexNumber *number*)
Sets the data at 'tagPath' in 'tagGroup' as a complex number.
¬†

void **TagGroupSetTagAsNumber**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a real number.
¬†
void **TagGroupSetTagAsRGBUInt16**(TagGroup *tagGroup*, String *tagPath*, Number *r*, Number *g*, Number *b*)
Sets the data at 'tagPath' in 'tagGroup' as a 16-bit rgb value.
¬†
void **TagGroupSetTagAsShort**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a short.
¬†
void **TagGroupSetTagAsShortPoint**(TagGroup *tagGroup*, String *tagPath*, Number *x*, Number *y*)
Sets the data at 'tagPath' in 'tagGroup' as a short point.
¬†
void **TagGroupSetTagAsShortRect**(TagGroup *tagGroup*, String *tagPath*, Number *t*, Number *l*, Number *b*, Number *r*)
Sets the data at 'tagPath' in 'tagGroup' as a short rect.
¬†
void **TagGroupSetTagAsString**(TagGroup *tagGroup*, String *tagPath*, String *s*)
Sets the data at 'tagPath' in 'tagGroup' as a string.
¬†
void **TagGroupSetTagAsTagGroup**(TagGroup *tagGroup*, String *tagPath*, TagGroup *subGroup*)
Sets the data at 'tagPath' in 'TagGroup' as a group.
¬†
void **TagGroupSetTagAsText**(TagGroup *tagGroup*, String *tagPath*, String *s*)
Sets the data at 'tagPath' in 'tagGroup' as a string.
¬†
void **TagGroupSetTagAsUInt16**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a 16-bit unsigned integer.
¬†
void **TagGroupSetTagAsUInt32**(TagGroup *tagGroup*, String *tagPath*, Number *number*)
Sets the data at 'tagPath' in 'tagGroup' as a 32-bit unsigned integer.
¬†
void **TagGroupSetTagRGBBitmap**(TagGroup *tagGroup*, String *tagPath*, ImageReference *image*)
Sets the data at 'tagPath' in 'tagGroup' as a RGB bitmap.
¬†
void **TagGroupWriteIndexedTagDataToStream**(TagGroup *tagGroup*, Number *index*, ScriptObject *stream*, Number *stream_endianness*)
Writes data from the indicated tag into the stream. The tag type determines how the data is written, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
void **TagGroupWriteTagDataToStream**(TagGroup *tagGroup*, String *tag_path*, ScriptObject *stream*, Number *stream_endianness*)
Reads data from the indicated tag into the stream. The tag type determines how the data is written, and 'stream_endianness' specifies the endianness of the stream, 1 == bigendian, 2 == littleendian.
¬†
Number **tan**(Number *v1*)
¬†
RealNumberExpression **tan**(RealNumberExpression)
Return the tangent of the number.
¬†
ComplexNumberExpression **tan**(ComplexNumberExpression)
Return the tangent of the number.
¬†
RealNumberExpression **tanh**(RealNumberExpression)
Return the hyperbolic tangent of the number.
¬†
Number **tanh**(Number *v1*)
¬†
ComplexNumberExpression **tanh**(ComplexNumberExpression)
Return the hyperbolic tangent of the number.
¬†
RealNumberExpression **tert**(RealNumberExpression *condition*, RealNumberExpression *truenumber*, RealNumberExpression *falsenumber*)
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
Number **tert**(Boolean *cond*, Number *trueval*, Number *falseval*)

¬†
ComplexNumberExpression **tert**(RealNumberExpression *condition*, ComplexNumberExpression *truenumber*, ComplexNumberExpression *falsenumber*)
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
RGBNumberExpression **tert**(RealNumberExpression *condition*, RGBNumberExpression *truenumber*, RGBNumberExpression *falsenumber*)
Evaluates the condition expression and returns either the truenumber or falsenumber expression depending on condition.
¬†
void **Test_AddTestAnnotation**(Component *c*, Number *kind*, Number *top*, Number *left*, Number *bottom*, Number *right*)
¬†
void **Test_AssignToImageRef**(ImageVariable *img*)
¬†
void **Test_AssignUniformRandom**(ImageReference *img*)
¬†
Number **Test_BananaEggs**(Number *one*, Number *two*, Number *three*)
¬†
void **Test_CallFunction_ImageParam**(Function *func*, ImageReference *img*)
¬†
ImageReference **Test_CallFunction_ImageReturn**(Function *func*)
¬†
void **Test_CallFunctionOnImage**(ImageReference *img*, String *sig*)
¬†
Function **Test_CompileScriptFunction**(String *sig*, String *script*)
¬†
Number **Test_ConvertDate**(String *date*)
¬†
Number **Test_CountGlobalObjects**(String *name*)
¬†
ImageReference **Test_CreateImageFromCPP**(String *name*, Number *type*, Number *x_size*, Number *y_size*)
¬†
void **Test_DoubleArg**(Number *d*)
¬†
void **Test_ForceImageUpdate**(ImageReference *img*)
¬†
void **Test_ForceWindowUpdate**(DocumentWindow *win*)
¬†
Function **Test_GetGlobalFunction**( )
¬†
void **Test_ImageReferenceCounting**( )
¬†
Number **Test_IncCounter**(Number *period*)
¬†
Boolean **Test_IsInBackground**( )
¬†
Function **Test_LookupPlugInFunction**(String *sig*)
¬†
void **Test_OperatorDelete**(Number)
¬†
Number **Test_OperatorNew**(Number)
¬†
TagGroup **Test_ReferenceCount_V1**(ROI *roi_out*)
¬†
TagGroup **Test_ReferenceCount_V2**(ROI *roi_out*)
¬†
ROI **Test_ReinterpretLongAsROIPtr**(ROI *r*)
¬†
void **Test_ResetCounter**( )
¬†
void **Test_SetGlobalFunction**(Function *func*)
¬†
Boolean **Test_SetTestFlag**(Number *index*, Boolean *val*)
¬†
Number **Test_SizeofData**(Number *index*)
¬†
void **Test_SurveyImage**(ImageReference *image*, ImageDisplay *imgDisp*, Number *survey_style*, Number

complex_mode, NumberVariable *min*, NumberVariable *max*)
¬†
void **Test_ThrowCPPException**(Number *index*)
¬†
Number **TextAnnotationGetAlignment**(Component *comp*)
Gets the alignment of the text in the text annotation.
¬†
void **TextAnnotationGetFixedPoint**(Component *comp*, NumberVariable *x*, NumberVariable *y*)
Gets the fixed point of the text annotation.
¬†
Number **TextAnnotationGetResizeStyle**(Component *comp*)
Gets the resize style of the text annotation.
¬†
String **TextAnnotationGetText**(Component *comp*)
Gets the text of a text annotation.
¬†
void **TextAnnotationSetAlignment**(Component *comp*, Number *alignment*)
Sets the alignment of the text in the text annotation.
¬†
void **TextAnnotationSetFixedPoint**(Component *comp*, Number *x*, Number *y*)
Sets the fixed point of the text annotation.
¬†
void **TextAnnotationSetResizeStyle**(Component *comp*, Number *style*)
Sets the resize style of the text annotation.
¬†
void **TextAnnotationSetText**(Component *comp*, String *text*)
Sets the text of a text annotation.
¬†
void **Throw**(Number *exception*)
Throws an exception by number.
¬†
void **Throw**(String *exception*)
Throws an exception by string.
¬†
Boolean **ThrowableIsValid**(Throwable *throw_*)
Returns true if 'throw_' is a valid object.
¬†
Throwable **ThrowableNullify**(!)
Assigns NULL to dst throwable.
¬†
Number **TickCount**( )
Return the MacOS system tick count.
¬†
RGBImageExpression **TimeBar**(String *title*, RGBImageExpression *expression*)
Displays a timebar with the given title during evaluation of the expression. Return the expression directly.
¬†
ComplexImageExpression **TimeBar**(String *title*, ComplexImageExpression *expression*)
Displays a timebar with the given title during evaluation of the expression. Return the expression directly.
¬†
RealImageExpression **TimeBar**(String *title*, RealImageExpression *expression*)
Displays a timebar with the given title during evaluation of the expression. Return the expression directly.
¬†
void **TransferTagsAndApplyDataBar**(ImageReference)
Transfer tags and apply data bar to the image.
¬†
void **TransformPointFromImageToWindow**(ImageReference *image*, Number *x_image*, Number *y_image*,
NumberVariable *x_window*, NumberVariable *y_window*)
Place in (x_window,y_window) the
¬†
RealNumberExpression **Trunc**(RealNumberExpression)
Return the number truncated to an integer (rounding towards zero).
¬†
Number **trunc**(Number *v1*)
¬†
void **trycatch**(Expression, Expression)

¬†

`void` **`tryrecover`**`( Expression, Expression )`

¬†

`ImageReference` **`TryToUse`**`( String` *`name`*`, Number` *`width`*`, Number` *`height`*`, Number` *`dataType`*`, Number` *`h`*`, Number` *`v`*`, NumberVariable` *`fresh`* `)`

Look for an image with the given name, width, height, dataType, and position [h,v]. Return it if found and create it if not. Store 1 into fresh if it was created and 0 if not.

¬†

`Boolean` **`TwoButtonDialog`**`( String` *`prompt`*`, String` *`acceptLabel`*`, String` *`rejectLabel`* `)`

Puts up a dialog with the given prompt and two buttons labeled according to the parameters. Returns 1 for the acceptLabel button and false for the other one.

¬†

`Number` **`unc`**`( String, Number` *`index`* `)`

Returns the unicode value of the first character of the string.

¬†

`RealNumberExpression` **`UniformRandom`**`( )`

Return a random number with uniform distribution between [0,1).

¬†

`void` **`UnregisterClass`**`( String` *`class_name`* `)`

¬†

`void` **`UnregisterCustomMenu`**`( Number` *`menuHandlerToken`* `)`

Unregister a custom menu. See the SDK documentation for more information.

¬†

`void` **`UnregisterObjectListener`**`( Number` *`object`*`, Number` *`id`* `)`

Remove object listener from OM object. See the SDK documentation for more information.

¬†

`void` **`UpdateDisplay`**`( ImageReference, Number` *`startRow`*`, Number` *`endRow`* `)`

Update the image display portion that displays the image rows from startRow to endRow.

¬†

`void` **`UpdateImage`**`( ImageReference )`

Update the image immediately.

¬†

`void` **`UpdatePictureAnnotation`**`( ImageReference, Number` *`annotationID`*`, ! PicHandle )`

Update the picture annotation indicated by the annotationID within the image with the new picHandle.

¬†

`void` **`UpdateTimeBarPercentage`**`( Number` *`percentage`* `)`

Updates an open time bar to the given percentage.

¬†

`Number` **`val`**`( String )`

Returns the numeric value of a string.

¬†

`Boolean` **`ValidAnnotation`**`( ImageReference, Number` *`annotationID`* `)`

Return 1 if the annotationID is valid; returns 0 otherwise.

¬†

`Number` **`variance`**`( ImageReference )`

Return the variance of the image.

¬†

`ComplexImageExpression` **`Warp`**`( ComplexImage` *`source`*`, RealImageExpression` *`sourceX`*`, RealImageExpression` *`sourceY`* `)`

Returns the bilinear interpolated value at the position [x,y] within the source image.

¬†

`RealImageExpression` **`Warp`**`( RealImage` *`source`*`, RealImageExpression` *`x`*`, RealImageExpression` *`y`* `)`

Returns the bilinear interpolated value at the position [x,y] within the source image.

¬†

`RGBImageExpression` **`Warp`**`( RGBImage` *`source`*`, RealImageExpression` *`sourceX`*`, RealImageExpression` *`sourceY`* `)`

Returns the bilinear interpolated value at the position [x,y] within the source image.

¬†

`void` **`while`**`( RealNumber, Expression )`

¬†

`void` **`WindowClose`**`( DocumentWindow` *`window`*`, Boolean` *`verify`* `)`

Closes the window, prompting the user if 'verify' is true.

¬†

`void` **`WindowGetContentBounds`**`( DocumentWindow` *`window`*`, NumberVariable` *`top`*`, NumberVariable` *`left`*`, NumberVariable` *`bottom`*`, NumberVariable` *`right`* `)`

Gets the bounding rectangle of the content area of the 'window'.
¬†
`void` **`WindowGetContentPosition`**`( DocumentWindow window, NumberVariable x, NumberVariable y )`
Gets the position of the top-left corner of the content area of the 'window'.
¬†
`void` **`WindowGetContentSize`**`( DocumentWindow window, NumberVariable x, NumberVariable y )`
Gets the size of the content area of the 'window'.
¬†
`void` **`WindowGetFrameBounds`**`( DocumentWindow window, NumberVariable top, NumberVariable left, NumberVariable bottom, NumberVariable right )`
Gets the bounding rectangle of the frame area of the 'window'.
¬†
`void` **`WindowGetFramePosition`**`( DocumentWindow window, NumberVariable x, NumberVariable y )`
Gets the position of the top-left corner of the frame area of the 'window'.
¬†
`void` **`WindowGetFrameSize`**`( DocumentWindow window, NumberVariable x, NumberVariable y )`
Gets the size of the frame area of the 'window'.
¬†
`void` **`WindowGetMousePosition`**`( DocumentWindow window, NumberVariable x, NumberVariable y )`
Gets the current position of the mouse in the windows coordinate system.
¬†
`String` **`WindowGetTitle`**`( DocumentWindow window )`
Gets the title of the window.
¬†
`Number` **`WindowGetType`**`( DocumentWindow window )`
Gets the type of thw window.
¬†
`void` **`WindowHide`**`( DocumentWindow window )`
Hides the window.
¬†
`Boolean` **`WindowIsOpen`**`( DocumentWindow window )`
Returns true if the window has not been closed.
¬†
`Boolean` **`WindowIsShown`**`( DocumentWindow window )`
Returns true if the window is shown.
¬†
`Boolean` **`WindowIsValid`**`( DocumentWindow window )`
Returns true if 'window' points to a valid object.
¬†
`DocumentWindow` **`WindowNullify`**`( ! )`
¬†
`void` **`WindowSelect`**`( DocumentWindow window )`
Brings 'window' to the front.
¬†
`void` **`WindowSendBehind`**`( DocumentWindow window, DocumentWindow behind_window )`
Sends 'window' behind 'behind_window'.
¬†
`void` **`WindowSetContentBounds`**`( DocumentWindow window, Number top, Number left, Number bottom, Number right )`
Sets the bounding rectangle of the content area of the 'window'.
¬†
`void` **`WindowSetContentPosition`**`( DocumentWindow window, Number x, Number y )`
Sets the position of the top-left corner of the content area of the 'window'.
¬†
`void` **`WindowSetContentSize`**`( DocumentWindow window, Number x, Number y )`
Sets the size of the content area of the 'window'.
¬†
`void` **`WindowSetFrameBounds`**`( DocumentWindow window, Number top, Number left, Number bottom, Number right )`
Sets the bounding rectangle of the frame area of the 'window'.
¬†
`void` **`WindowSetFramePosition`**`( DocumentWindow window, Number x, Number y )`
Sets the position of the top-left corner of the frame area of the 'window'.
¬†
`void` **`WindowSetFrameSize`**`( DocumentWindow window, Number x, Number y )`

Sets the size of the frame area of the 'window'.
¬†

`void` **`WindowSetTitle`**`( DocumentWindow` *`window`*`, String` *`title`*` )`
Sets the title of the window.
¬†

`void` **`WindowShow`**`( DocumentWindow` *`window`*` )`
Shows the window.
¬†

`void` **`WindowUpdate`**`( DocumentWindow` *`window`*` )`
Updates 'window's display.
¬†

`void` **`WriteFile`**`( Number` *`file`*`, Number` *`encoding`*`, String` *`data`*` )`
Write the string to the file with the specified encoding.
¬†

`void` **`WriteFile`**`( Number` *`file`*`, String` *`data`*` )`
Write the string to the file.
¬†

`void` **`WriteRawStream`**`( Number` *`rawStream`*`, Number` *`data`*`, Number` *`length`*` )`
Write length bytes from the memory pointed to by data to rawStream.
¬†

`Number` **`XX_GetProcessKey`**`( Number` *`pid`*` )`
Return a keystroke associated with the background process indicated by pid.
¬†

`void` **`XX_InstallImageProcess`**`( ImageReference, Number` *`pid`*` )`
Associate the process indicated by pid with the image. Keystrokes going to the image will queue in the process after this call.
¬†

`void` **`Yield`**`( )`
Yield to another background task.
¬†

Core Dialog Library Functions

Return to Top

Dialog Attribute Access Functions

Return to Core Dialog Library Functions
Return to Top

Set Property Values

TagGroup **DLGSide**(TagGroup element, String side)
Sets the 'Side' attribute: "Left", "Right", "Top", "Bottom", "Center"; returns the element.

TagGroup **DLGAnchor**(TagGroup element, String anchor)
Sets the 'Anchor' attribute: "North", "South", "East", "West"; returns the element.

TagGroup **DLGFill**(TagGroup element, String fill)
Sets the 'Fill' attribute: "X", "Y", "XY", "Both"; returns the element.

TagGroup **DLGExpand**(TagGroup element, Number do_expand)
Sets the 'Expand' attribute: 0 for x-direction, 1 for y-direction; returns the element.

TagGroup **DLGExpand**(TagGroup element, String s)
Sets the 'Expand' attribute: "X", "Y", "XY"; returns the element.

TagGroup **DLGInternalPadding**(TagGroup element, number x_pad, number y_pad)
Sets the 'InternalPadding' attribute; returns the element.

TagGroup **DLGExternalPadding**(TagGroup element, Number x_pad, Number y_pad)
Sets the 'ExternalPadding' attribute; returns the element.

TagGroup **DLGValue**(TagGroup element, Number value)
Sets the numeric 'Value' attribute; returns the element.

TagGroup **DLGValue**(TagGroup element, string value)

Sets the string 'Value' attribute; returns the element.

TagGroup **DLGValue**(object frame, string identifier, number val)
Sets the numeric 'Value' attribute in a labeled element in a uiframe object; returns the TagGroup containing the identifier.

TagGroup **DLGValue**(object frame, string identifier, string val)
Sets the string 'Value' attribute in a labeled element in a uiframe object; returns the TagGroup containing the identifier.

TagGroup **DLGMinValue**(TagGroup element, Number value)
Sets the 'MinValue' attribute; returns the element.

TagGroup **DLGMaxValue**(TagGroup element, Number value)
Sets the 'MaxValue' attribute; returns the element.

TagGroup **DLGIdentifier**(TagGroup element, String id)
Sets the 'Identifier' attribute; returns the element.

TagGroup **DLGWidth**(TagGroup element, number width)
Sets the 'Width' attribute; returns the element.

TagGroup **DLGHeight**(TagGroup element, number height)
Sets the 'Height' attribute; returns the element.

TagGroup **DLGFont**(TagGroup element, string font)
Sets the 'Font' attribute; returns the element.

TagGroup **DLGTitle**(TagGroup element, string title)
Sets the 'Title' attribute; returns the element.

TagGroup **DLGActionMethod**(TagGroup element, string action_method)
Sets the 'ActionMethod' attribute; returns the element.

TagGroup **DLGChangedMethod**(TagGroup element, string changed_method)
Sets the 'ChangedMethod' attribute; returns the element.

TagGroup **DLGMultipleSelect**(TagGroup tg, number multiple_select)
Sets the 'MultipleSelect' attribute for Lists; returns the element.

TagGroup **DLGLayout**(TagGroup tg, TagGroup layout)
Sets the 'Layout' attribute for Groups; returns the element.

TagGroup **DLGBitmapData**(TagGroup tg, Image img)
Sets the image data for a bitmap

TagGroup **DLGInvalid**(TagGroup tg, number isInvalid)
Sets the 'Invalid' attribute for a modified element. Used for run-time update of dialog displays,

TagGroup **DLGLabel**(TagGroup tg, string label)
Sets the 'Label' attribute for an element

Get Property Values

number **DLGGetValue**(TagGroup tags, number &value)

Gets the numeric 'Value' attribute setting; returns 1 if succeeded, 0 otherwise.

number **DLGGetValue**(TagGroup tags, string &value)
Gets the string 'Value' attribute setting; returns 1 if succeeded, 0 otherwise.

number **DLGGetValue**(object frame, string identifier, number &val)
Gets the numeric 'Value' attribute setting for a labeled element in a uiframe object; returns 1 if succeeded, 0 otherwise.

number **DLGGetValue**(object frame, string identifier, string &val)
Gets the string 'Value' attribute setting for a labeled element identifier in a uiframe object; returns 1 if succeeded, 0 otherwise.

number **DLGGetValue**(TagGroup tags)
Returns the numeric 'Value' attribute setting.

string **DLGGetStringValue**(TagGroup tags)
Returns the string 'Value' attribute setting.

TagGroup **DLGGetItems**(TagGroup tags)
Returns the TagGroup in the 'Items' attribute of an element.

number **DLGGetIdentifier**(TagGroup tags, string &identifier)
Returns the string in the 'Identifier' attribute of an element.

number **DLGGetNthLabel**(TagGroup tags, number index, string &label_str)
Gets the label of the nth item in the 'Items' TagGroup of an element; returns 1 if succeeded, 0 otherwise.

string **DLGGetType**(TagGroup tags)
Gets the type attribute of an element.

number **DLGGetWidth**(TagGroup tags)
Gets the width of an element

number **DLGGetHeight**(TagGroup tags)
Gets the height of an element

Image **DLGGetBitmapData**(TagGroup bitmap)
Gets the RGB image stored in a bitmap

String **DLGGetLabel**(TagGroup element)
Gets the label of an element

String **DLGGetTitle**(TagGroup element)
Gets the label of an element (dialog or label)

Dialog Construction Functions

Return to Core Dialog Library Functions

Return to Top

TagGroup **DLGCreateElement**(string type)
Create a dialog element of the specified type; returns the newly created element.

TagGroup **DLGCreateElementWithItems**(string type, TagGroup &items)
Create a container of the specified type, and passes back an empty items list; returns the container.

TagGroup **DLGCreateElementWithItems**(string type)
Create a container of the specified type

TagGroup **DLGAddElement**(TagGroup container, TagGroup element)
Add an element to a container or element list; returns the element.

TagGroup **DLGAddElement**(TagGroup container, TagGroup element, String side)
Add an element with the specified 'Side' attribute to a container or element list; returns the element.

TagGroup **DLGAddElement**(TagGroup container, TagGroup element, String side, String anchor)
Add an element with the specified 'Side' and 'Anchor' attributes to a container or element list; returns the element.

TagGroup **DLGAddElementBefore**(TagGroup container, number index, TagGroup element)
Insert an element before the given index; returns the element

TagGroup **DLGAddElementAfter**(TagGroup container, number index, TagGroup element)
Insert an element after the given index; returns the element

TagGroup **DLGGetElement**(TagGroup container, number index)
Retrieve an element at the given index from a container.

TagGroup **DLGGetElement**(TagGroup container, string label, number &index)
Retrieve an element with the given label; passes back the index of the element

TagGroup **DLGGetElement**(TagGroup container, string label)
Retrieve an element with the given label

TagGroup **DLGRemoveElement**(TagGroup container, number index)
Remove an element at the given index from a container; returns the removed element.

TagGroup **DLGRemoveLastElement**(TagGroup container)
Remove the last element from a container; returns the removed element.

TagGroup **DLGRemoveElement**(TagGroup container, string label)
Remove an element with the given label; returns the removed element.

Number **DLGSetElement**(TagGroup container, number index, TagGroup new_element)
Sets the element at the given index; returns 1 if succeeded, 0 otherwise

Number **DLGSetElement**(TagGroup container, string label, TagGroup new_element)
Sets the element with the given label; returns 1 if succeeded, 0 otherwise

TagGroup **DLGSetElementLabel**(TagGroup container, number index, string new_label)
Sets the label of the element at the given index; returns the element

TagGroup **DLGSetElementLabel**(TagGroup container, string label, string new_label)
Sets the label of the element with the given label; returns the element

TagGroup **DLGCreateDialog**(string title, TagGroup &dialog_items)
Create a dialog container with the given title; passes back an empty items list, and returns the dialog container.

TagGroup **DLGCreateDialog**(string title)
Create a dialog container with the given title

TagGroup **DLGCreateTableLayout**(number cols, number rows, number uniform)
Create a table layout with the given number of columns, rows, and whether the table cells are uniform in size; returns the table layout.

TagGroup **DLGCreateTableLayout**(number cols, number rows, string uniform)
Create a table layout with the given number of columns, rows, and whether the table cells are uniform in size; returns the table layout.

TagGroup **DLGTableLayout**(TagGroup element, number columns, number rows, number uniform)
Sets the 'Layout' attribute to a table layout with the given number of columns, rows, and cell uniformity flag; returns the element.

TagGroup **DLGTableLayout**(TagGroup tg, number columns, number rows, string uniform_str)
Sets the 'Layout' attribute to a table layout with the given number of columns, rows, and cell uniformity flag; returns the element.

Modeless Positioning Utilities

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGBuildPositionFromApplication**()
Creates a dialog position descriptor that uses the bounds of the largest document window as reference; returns the position descriptor.

TagGroup **DLGBuildPositionFromWindow**(DocumentWindow wnd)
Creates a dialog position descriptor that uses the bounds of the given window as reference; returns the position descriptor.

TagGroup **DLGBuildPositionFromUIFrame**(object frame)
Creates a dialog position descriptor that uses the bounds of the UI frame as reference; returns the position descriptor.

TagGroup **DLGBuildAutoSize**()
Creates an auto-size position descriptor; returns the descriptor.

TagGroup **DLGBuildMatchSize**()
Creates a match-size position descriptor; returns the descriptor.

TagGroup **DLGBuildAbsoluteSize**(number inches)
Creates an absolute-size position descriptor; returns the descriptor.

TagGroup **DLGBuildRelativePosition**(string sense, number rel)
Creates a relative-size position descriptor; sense: "Inside", "Outside"; rel: 0, 1; returns the descriptor.

TagGroup **DLGBuildPosition**(number top, number left, number bottom, number right)
Creates a dialog position descriptor with the given reference bounds; returns the position descriptor.

TagGroup **DLGPosition**(TagGroup tags, TagGroup position)
Sets the 'Positioning' attribute of a dialog to the given position descriptor; returns the dialog.

TagGroup **DLGPosition**(TagGroup tags, number top, number left, number bottom, number right)
Creates a dialog position descriptor with the given reference bounds, and adds it to the given dialog tags; returns the dialog tags.

Bitmap Functions For Bevel Buttons

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateBitmap**(Image img)
Creates a bitmap TagGroup from the given image.

RGBImage **DLGMakeRaised**(image im)
Creates an RGBImage with depressed borders from the given image.

RGBImage **DLGMakeLowered**(image im)
Creates an RGBImage with raised borders from the given image.

Label Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateLabel**(String label_string)
Creates a label with the given text.

TagGroup **DLGCreateLabel**(string label_string, number width)
Creates a label with the given text and maximum number of characters.

Button Functions

Return to Core Dialog Library Functions

Return to Top

TagGroup **DLGCreatePushButton**(String title, String action_method)
Creates a single-state text button with the given display title and action method.

TagGroup **DLGCreateBevelButton**(TagGroup onBitmap, TagGroup offBitmap, string actionmethod)
Creates a single-state bitmap button with the given on/off bitmap TagGroups and action method.

TagGroup **DLGCreateBevelButton**(Image onImage, Image offImage, string actionmethod)
Creates a single-state bitmap button with the given on/off images and action method.

TagGroup **DLGCreateBevelButton**(Image img, string actionmethod)
Creates a single-state bitmap button with the given image and action method.

TagGroup **DLGCreateDualStateBevelButton**(string identifier, TagGroup onBitmap, TagGroup offBitmap, string actionmethod)
Creates a dual-state bitmap button with the given identifier, on/off bitmap TagGroups, and action method.

TagGroup **DLGCreateDualStateBevelButton**(string identifier, Image onImage, Image offImage, string actionmethod)
Creates a dual-state bitmap button with the given identifier, on/off images, and action method.

TagGroup **DLGCreateDualStateBevelButton**(string identifier, Image img, string actionmethod)
Creates a dual-state bitmap button with the given identifier, image, and action method.
=> "action method" has to be defined in the *class* and be of syntax: void actionmethod(**object** *self*)

Number **DLGGetBevelButtonOn**(object frame, String identifier, TagGroup &tags)
Gets the On/Off state of a labeled bevel button in a uiframe object; returns the button TagGroup in tags, and 1 if succeeded, 0 otherwise.

TagGroup **DLGBevelButtonOn**(TagGroup button_tags, number isOn)
Sets the On/Off state of a button; returns the button TagGroup.

TagGroup **DLGBevelButtonOn**(object frame, string identifier, number isOn)
Sets the On/Off state of a labeled button in a uiframe object; returns the button TagGroup.

Group Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateGroup**()
Creates a group container

TagGroup **DLGCreateGroup**(TagGroup &items)
Creates a group container; passes back an empty items list, and returns the container.

TagGroup **DLGGroupItems**(TagGroup item1, TagGroup item2)
Creates a group containing the given two items; returns the container.

TagGroup **DLGGroupItems**(TagGroup item1, TagGroup item2, TagGroup item3)
Creates a group containing the given three items; returns the container.

TagGroup **DLGGroupItems**(TagGroup item1, TagGroup item2, TagGroup item3, TagGroup item4)
Creates a group containing the given four items; returns the container.

Field Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateRealField**(number value)
Creates a real-valued field with the given initial value, field width, and format.

TagGroup **DLGCreateRealField**(number value, number width, number format)
Creates a real-valued field with the given initial value, field width, and format.

TagGroup **DLGCreateRealField**(number value, number width, number format, string changedmethod)
Creates a real-valued field with the given initial value, field width, format, and changed method.

TagGroup **DLGCreateIntegerField**(number value)
Creates a integer-valued field with the given initial value, and field width.

TagGroup **DLGCreateIntegerField**(number value, number width)
Creates a integer-valued field with the given initial value, and field width.

TagGroup **DLGCreateIntegerField**(number value, number width, string changedmethod)
Creates a integer-valued field with the given initial value, field width, and changed method.

TagGroup **DLGCreateStringField**(String value)
Creates a string-valued field with the given initial value, and field width.

TagGroup **DLGCreateStringField**(String value, number width)
Creates a string-valued field with the given initial value, and field width.

TagGroup **DLGCreateStringField**(String value, number width, string changedmethod)
Creates a string-valued field with the given initial value, field width, and changed method.

Checkbox Functions

Return to Core Dialog Library Functions

Return to Top

TagGroup **DLGCreateCheckBox**(string title)
Creates a checkbox with the given text.

TagGroup **DLGCreateCheckBox**(string title, number state)
Creates a checkbox with the given text and initial checked/unchecked state

TagGroup **DLGCreateCheckBox**(string title, number state, string changedmethod)
Creates a checkbox with the given text, initial checked/unchecked state, and changed method.

=> "changedmethod" has to be defined in the *class* and be of syntax: void changedmethod(**object** *self*, **taggroup** tg)

Box Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateBox**()
Creates a box container

TagGroup **DLGCreateBox**(String title)
Creates a box with the given title

TagGroup **DLGCreateBox**(TagGroup &items)
Creates a box container; passes back an empty items list, and returns the container.

TagGroup **DLGCreateBox**(String title, TagGroup &items)
Creates a box with the given title; passes back an empty items list, and returns the container.

Radio Button Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateRadioList**( )
Creates a radio button list container

TagGroup **DLGCreateRadioList**(number value)
Creates a radio button list container with the given initial value

TagGroup **DLGCreateRadioList**(number value, string changedmethod)
Creates a radio button list container with the given initial value and changed method;

TagGroup **DLGCreateRadioList**(TagGroup &items)

Creates a radio button list container; passes back an empty items list, and returns the container.

TagGroup **DLGCreateRadioList**(TagGroup &items, number value)
Creates a radio button list container with the given initial value; passes back an empty items list, and returns the container.

TagGroup **DLGCreateRadioList**(TagGroup &items, number value, string changedmethod)
Creates a radio button list container with the given initial value and changed method; passes back an empty items list, and returns the container.

TagGroup **DLGCreateRadioItem**(string label, number value)
Createsa radio button item with the given title and value

TagGroup **DLGAddRadioItem**(TagGroup rad_container, string label, number value)
Creates and adds a radio button item with the given title and value to a radio button list; returns the radio button item.

List Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateList**(number width, number height)
Creates a list container with the given width and height

TagGroup **DLGCreateList**(string changedmethod, number width, number height)
Creates a list container with the given width, height, and changed method;

TagGroup **DLGCreateList**(string changedmethod, string actionmethod, number width, number height)
Creates a list container with the given width, height, changed method and action method;

TagGroup **DLGCreateList**(TagGroup &items, number width, number height)
Creates a list container with the given width and height; passes back an empty items list, and returns the container.

TagGroup **DLGCreateList**(TagGroup &items, string changedmethod, number width, number height)
Creates a list container with the given width, height, and changed method; passes back an empty items list, and returns the container.

TagGroup **DLGCreateList**(TagGroup &items, string changedmethod, string actionmethod, number width, number height)
Creates a list container with the given width, height, changed method and action method; passes back an empty items list, and returns the container.

TagGroup **DLGCreateListItem**(string label, number selected)
Creates a list item with the given label.

TagGroup **DLGAddListItem**(TagGroup item_container, string label, number selected)
Adds a list item with the given label to a list.

Popup Menu Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreatePopup**( )
Creates a popup menu

TagGroup **DLGCreatePopup**(number value)
Creates a popup menu with the given initial value

TagGroup **DLGCreatePopup**(number value, string changedmethod)
Creates a popup menu with the given initial value and changed method

TagGroup **DLGCreatePopup**(TagGroup &items)
Creates a popup menu; passes back an empty items list, and returns the container..

TagGroup **DLGCreatePopup**(TagGroup &items, number value)
Creates a popup menu with the given initial valuepasses back an empty items list, and returns the container.

TagGroup **DLGCreatePopup**(TagGroup &items, number value, string changedmethod)
Creates a popup menu with the given initial value and changed method; passes back an empty items list, and returns the container.

TagGroup **DLGCreatePopup**(string title, TagGroup &label_item, TagGroup &popup_item, TagGroup &choice_items, number value)
Creates a named popup menu group with the given title and initial value; passes back the label for the popup title, the popup menu, and an empty popup items list; returns the named popup group.

TagGroup **DLGCreateBoxedPopup**(string boxTitle, TagGroup &popup_item, TagGroup &choice_items, number value)
Creates a named box containing a popup menu with the given title and initial value; passes back the popup menu and an empty popup items list; returns the box element.

TagGroup **DLGCreatePopupItemEntry**(string label)
Creates a popup entry with the given label.

TagGroup **DLGAddPopupItemEntry**(TagGroup container, string label)
Adds a popup entry with the given label to a list of popup items; returns the popup entry.

Choice Item Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateChoice**( )
Creates a choice item

TagGroup **DLGCreateChoice**(number value)
Creates a choice item with the given initial value

TagGroup **DLGCreateChoice**(number value, string changedmethod)
Creates a choice item with the given initial value and changed method

TagGroup **DLGCreateChoice**(TagGroup &items)
Creates a choice item; passes back an empty items list, and returns the container..

TagGroup **DLGCreateChoice**(TagGroup &items, number value)
Creates a choice item with the given initial valuepasses back an empty items list, and returns the container.

TagGroup **DLGCreateChoice**(TagGroup &items, number value, string changedmethod)
Creates a choice item with the given initial value and changed method; passes back an empty items list, and returns the container.

TagGroup **DLGCreateChoice**(string title, TagGroup &label_item, TagGroup &choice_item, TagGroup &choice_items, number value)
Creates a named choice item group with the given title and initial value; passes back the label for the choice title, the choice item, and an empty choice items list; returns the named choice group.

TagGroup **DLGCreateBoxedChoice**(string boxTitle, TagGroup &choice_item, TagGroup &choice_items, number value)
Creates a named box containing a choice item with the given title and initial value; passes back the choice item and an empty choice items list; returns the box element.

TagGroup **DLGCreateChoiceItemEntry**(string label)
Creates a choice entry with the given label

TagGroup **DLGAddChoiceItemEntry**(TagGroup container, string label)
Adds a choice entry with the given label to a list of choice items; returns the choice entry.

Panel Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreatePanelList**( )
Creates a panel list container

TagGroup **DLGCreatePanelList**(number value)
Creates a panel list container with the given initially selected panel index;

TagGroup **DLGCreatePanelList**(TagGroup &items)
Creates a panel list container; passes back an empty panel list, and returns the container.

TagGroup **DLGCreatePanelList**(TagGroup &items, number value)
Creates a panel list container with the given initially selected panel index; passes back an empty panel list, and returns

the container.

TagGroup **DLGCreatePanel**( )
Creates a panel element

TagGroup **DLGCreatePanel**(TagGroup &items)
Creates a panel element; passes back an empty items list, and returns the element.

TagGroup **DLGAddPanel**(TagGroup container, TagGroup panel)
Adds a panel element to a panel list; returns the panel element.

TagGroup **DLGAddPanel**(TagGroup container)
Creates and adds a panel element to a panel list; returns an empty items list for the new panel.

Tab Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateTabList**( )
Creates a tab list container

TagGroup **DLGCreateTabList**(number value)
Creates a tab list container with the given initially selected tab index

TagGroup **DLGCreateTabList**(number value, string changedmethod)
Creates a tab list container with the given initially selected tab index and changed method

TagGroup **DLGCreateTabList**(TagGroup &items)
Creates a tab list container; passes back an empty tab list, and returns the container.

TagGroup **DLGCreateTabList**(TagGroup &items, number value)
Creates a tab list container with the given initially selected tab index; passes back an empty tab list, and returns the container.

TagGroup **DLGCreateTabList**(TagGroup &items, number value, string changedmethod)
Creates a tab list container with the given initially selected tab index and changed method; passes back an empty tab list, and returns the container.

TagGroup **DLGCreateTab**(string label)
Creates a tab with the given label

TagGroup **DLGCreateTab**(TagGroup &tab_items, string label)
Creates a tab with the given label; passes back an empty panel items list and returns the tab element.

TagGroup **DLGAddTab**(TagGroup container, string label)
Creates and adds a tab with the given label to a tab list; returns an empty items list for the new tab.

TagGroup **DLGAddTab**(TagGroup container, TagGroup tab)

Adds a tab to a tab list; returns the tab element.

Text Box Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateTextBox**(number width, number height, number length)
Creates a text box with the given width, height, and text length.

Image Popup Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateImagePopup**()
Creates a popup menu listing the currently open images.

TagGroup **DLGCreateImagePopup**(number image_id)
Creates a popup menu listing the currently open images, with the name of the image with the given image ID initially selected.

Progress Bar Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateProgressBar**(string identifier)
Creates a progress bar.

TagGroup **DLGSetProgress**(object frame, string identifier, number percentage_complete)
Sets the progress status of a progress bar in a uiframe object; returns the TagGroup containing the identifier.

TagGroup **DLGGetProgress**(object frame, string identifier, number &percentage_complete)
Gets the progress status of a progress bar in a uiframe object; returns the TagGroup containing the identifier.

Graphic Functions

Return to Core Dialog Library Functions
Return to Top

TagGroup **DLGCreateGraphic**(number width, number height)
Creates a graphic element with the given width and height

TagGroup **DLGAddBitmap**(TagGroup graphic, TagGroup bitmap_item)
Adds a bitmap to a graphic element

TagGroup **DLGAddBitmap**(TagGroup graphic, Image img)
Creates a bitmap from the given image and adds it to the graphic element