MultiGen® II API Reference Manual

November 1996

550 S. Winchester Blvd., Suite 500, San Jose, CA 95128 Phone (408) 261-4100 Fax (408) 261-4101

MultiGen® II API Reference Guide, November 1996

Printed in the U.S.A.

Part Number: MG-????

Contents

mgNewTexture

NAME

mgNewTexture - creates a new entry in a database's texture palette.

SYNOPSIS

#include lapistruc.h

mgbool mgNewTexture (mgrec *db, int index, unsigned char *texels, char *name, mgrec *rec, int x, int y);

DESCRIPTION

Given a database node *db*, a texture palette index *index*, an array of texels *texels*, a texture name *name*, a record pointer *rec*, and a palette positiion *x y*, **mgNewTexture** creates a new texture in the database's texture palette with the name *name* and at the position (*x, y*) with the given *texels*. A pointer to the texture record is returned if creation is successful, otherwise **mgNULL** is returned.

ARGUMENTS

db	the database node.
index	the index of the new texture palette entry.
texels	the array of texels that make up the texture image.
name	the name of the texture.
rec	the returned texture record pointer.
x	the X position in the texture palette.
y	the Y position in the texture palette.

RETURNS

rec	the new texture record.
mgbool	returns **mgTRUE** if texture creation is successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgCopyTexture, **mgDeleteTexture**, **mgReplaceTexture**.

COPYRIGHT

mgCopyTexture

NAME

mgCopyTexture - makes a copy of a texture palette entry.

SYNOPSIS

#include lapitexture.h

int mgCopyTexture (mgrec *dstDb, mgrec *srcDb, char *newTextureName, int srcIndex);

DESCRIPTION

mgCopyTexture makes a new texture palette entry from a source database, *srcDb*, and index, *srcIndex*, and loads it in the destination database's texture palette using the first available index and the given *newTextureName*. It returns the index of the new entry. The texture can be copied from one database to another, or it can be copied to a new index in the same database by specifying the same database for *dstDb* and srcDb.

ARGUMENTS

dstDb the database to which the texture is to be copied.

srcDb the database from which the texture is to be copied.

newTextureName

the name of the new entry (the copy).

srcIndex the index of the texture palette entry to be copied.

RETURNS

int returns the index of the copy if the texture was successfully copied; otherwise returns **-1**.

ACCESS LEVEL

Level 2

SEE ALSO

mgCopyTexturePalette.

COPYRIGHT

mgCopyTexturePalette

NAME

mgCopyTexturePalette - copies a texture palette from one database to another.

SYNOPSIS

#include lapitexture.h

mgbool mgCopyTexturePalette (mgrec *dstDb, mgrec *srcDb)

DESCRIPTION

mgCopyTexturePalette copies the texture palette of one database (*srcDb*) to another database (*dstDb*).

ARGUMENTS

dstDb	the database to copy the palette to.
srcDb	the database to copy the palette from.

RETURNS

mgbool	returns **mgTRUE** if the copy was successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgCopyTexture.

COPYRIGHT

mgDeleteTexture

NAME

mgDeleteTexture - deletes an entry from a database's texture palette.

SYNOPSIS

#include lapitexture.h

void mgDeleteTexture (mgrec *db, int index);

DESCRIPTION

mgDeleteTexture deletes a texture palette entry, specified by its *index*, from the given database's texture palette.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry to delete.

ACCESS LEVEL

Level 2

SEE ALSO

mgReplaceTexture

COPYRIGHT

mgFlipImage

NAME

mgFlipImage - flips an image about its u-axis.

SYNOPSIS

#include lapicolor.h

int mgFlipImage (unsigned char *pixels, int width, int height, int type);

DESCRIPTION

mgFlipImage flips an image about its u-axis. This is useful for converting an image with an origin in the upper-left corner to one with an origin in the lower-left corner (the orientation used by MultiGen, IRIS GL and OpenGL). **mgFlipImage** can also be used to perform the opposite conversion.

ARGUMENTS

pixels	a pointer to the image pixels.
width	the width of the image.
height	the height of the image.
type	the image type.
	0 = Not Used.
	1 = Not Used.
	2 = Intensity (1 channel)
	3 = Intensity-Alpha (2 channels)
	4 = RGB (3 channels)
	5 = RGBA (4 channels)

RETURNS

int	returns a status code**:**
	TXTIO_NO_ERROR = 0 = Success.
	TXTIO_MALLOC_ERR = -1 = Memory Allocation Failure.
pixels	returns the image pixels.

ACCESS LEVEL

Level 2

SEE ALSO

mgReadImage, **mgWriteImage**.

COPYRIGHT

mgGetFirstTexture
mgGetNextTexture

NAME

mgGetFirstTexture - gets the first texture in a database's palette.

mgGetNextTexture - gets successive textures in a database's palette.

SYNOPSIS

#include lapitexture.h

mgbool mgGetFirstTexture (mgrec* db, int *index, char *textureName ;

mgbool mgGetNextTexture (mgrec* db, int *index, char *textureName ;

DESCRIPTION

mgGetFirstTexture gets the index and name of the first texture in a database's palette. **mgGetNextTexture** gets the indices and names of the remainder of the textures in the palette after **mgGetFirstTexture** is used.

EXAMPLE

```
int index;
mgrec *db;
char name [ 200 ];
if ( mgGetFirstTexture ( db, &index, name ) ) {
  while ( mgGetNextTexture ( db, &index, name ) ) {
  }
}
```

ARGUMENTS

db	a database record.
index	the address of the index variable.
name	the name of the texture.

RETURNS

mgbool	returns **mgTRUE** if a texture is found; otherwise returns **mgFALSE**. **mgFALSE** should be used to flag the end of the traversal.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureName, **mgGetTextureIndex**.

COPYRIGHT

mgGetTextureAttributes

NAME

mgGetTextureAttributes - gets the attributes record for a texture palette entry.

SYNOPSIS

#include lapitexture.h

mgbool mgGetTextureAttributes (mgrec* db, int index, mgrec *rec);

DESCRIPTION

mgGetTextureAttributes gets the attributes record for an entry specified by its *index* in the given database's texture palette.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.
rec	a pointer to the attributes record.

RETURNS

mgbool	returns **mgTRUE** if the attributes are found; otherwise returns **mgFALSE.**
rec	returns the attributes record.

ACCESS LEVEL

Level 1

SEE ALSO

mgSetTextureAttributes.

COPYRIGHT

mgGetTextureCount

NAME

mgGetTextureCount - gets the number of entries in a database's texture palette.

SYNOPSIS

#include lapitexture.h

int mgGetTextureCount (mgrec* db);

DESCRIPTION

mgGetTextureCount returns the number of entries in a database's texture palette.

ARGUMENTS

db a database node.

RETURNS

int returns the number of entries in the texture palette, or **-1** on failure**.**

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureSize, **mgGetTextureTotalSize**.

COPYRIGHT

mgGetTextureIndex

NAME

mgGetTextureIndex - Gets the index of a texture palette entry from its name.

SYNOPSIS

#include lapitexture.h

int mgGetTextureIndex (mgrec* db, char* name);

DESCRIPTION

mgGetTextureIndex returns the index of a texture palette entry with the given *name* from the given database's texture palette.

ARGUMENTS

db	a database node.
name	the name of the texture palette entry.

RETURNS

int	returns the index of the texture palette entry, or returns **-1** if no entry with the specified name is found in the specified database's texture palette.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureName, **mgGetFirstTexture**, **mgGetNextTextureInPalette**.

COPYRIGHT

mgGetTextureName

NAME

mgGetTextureName - Gets the name of a texture palette entry.

SYNOPSIS

#include lapitexture.h

char * mgGetTextureName (mgrec* db, int index);

DESCRIPTION

mgGetTextureName returns the name of an entry from a given database's texture palette. NOTE: This function returns a pointer to the actual string. This pointer should **NOT** be freed.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.

RETURNS

char*	returns the name of the texture, or returns **mgNULL** if a texture with the specified index is not found in the database's texture palette.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureIndex, **mgSetTextureName**, **mgGetFirstTexture**, **mgGetNextTexture**.

COPYRIGHT

mgGetTexturePosition

NAME

mgGetTexturePosition - gets the position of a texture palette entry.

SYNOPSIS

#include lapitexture.h

mgbool mgGetTexturePosition (mgrec* db, int index, int *x, int *y);

DESCRIPTION

mgGetTexturePosition gets the position of the lower left corner (as displayed in MultiGen) of an entry in a given database's texture palette. The position is relative to the lower left corner of the palette.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.
x, y	pointers to the x & y variables. x & y are filled with the lower left corner of the image

RETURNS

mgbool	returns **mgTRUE** on success and if the texture was found in the database's palette; otherwise returns **mgFALSE.**
x, y	the x & y position of the lower left corner of the image.

ACCESS LEVEL

Level 1

SEE ALSO

mgSetTexturePosition.

COPYRIGHT

mgGetTextureSize

NAME

mgGetTextureSize - gets the size of a texture.

SYNOPSIS

#include lapitexture.h

int mgGetTextureSize (mgrec* db, int index);

DESCRIPTION

mgGetTextureSize returns the size (in bytes) of the image from one of the texture palette entries of a given database.

ARGUMENTS

db	a database node.
index	the index of the texture.

RETURNS

int	returns the size of the texture, if the texture was found in the database's palette; otherwise returns **-1**.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureCount, **mgGetTextureTotalSize**.

COPYRIGHT

mgGetTextureTexels

NAME

mgGetTextureTexels - gets a texture palette entry's texels.

SYNOPSIS

#include lapitexture.h

unsigned char * mgGetTextureTexels (mgrec* db, int index);

DESCRIPTION

Given a database node, *db*, and a texture palette entry defined by *index*, **mgGetTextureTexels** gets the texels for an entry from the database's texture palette. This function returns a pointer to the texels in memory. NOTE: This function returns a pointer to the actual texels. This pointer should **NOT** be freed.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.

RETURNS

mgbool	returns a pointer to the entry's texels; or on failure, or if the specified entry cannot be found, returns **mgNULL**.

ACCESS LEVEL

Level 1

SEE ALSO

mgSetTextureTexels.

COPYRIGHT

mgGetTextureTotalSize

NAME

mgGetTextureTotalSize - Gets the total memory usage (size in bytes) of a database's textures.

SYNOPSIS

#include lapitexture.h

int mgGetTextureTotalSize (mgrec* db);

DESCRIPTION

mgGetTextureTotalSize returns the total size, in bytes, of the images from all of the given database's texture palette entries.

ARGUMENTS

db a database node.

RETURNS

int returns the size in bytes; or returns or returns **-1** on failure.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureCount, **mgGetTextureSize**.

COPYRIGHT

mgInsertTexture

NAME

mgInsertTexture - loads a texture and creates a new entry in a database's texture palette.

SYNOPSIS

#include lapitexture.h

int mgInsertTexture (mgrec* db, char *filename);

DESCRIPTION

mgInsertTexture loads a texture and creates a new entry in a given database's texture palette. This function automatically positions the texture in the texture palette such that no other textures are covered, and assigns an index to the texture. The index of the new texture is returned.

ARGUMENTS

db	a database node.
filename	the name of the texture file to load.

RETURNS

int	returns the assigned index of the new entry when the texture is inserted successfully; otherwise returns **-1**.

ACCESS LEVEL

Level 1

SEE ALSO

mgReadTexture, **mgInsertTextureAndAlpha**, **mgReadTextureAndAlpha**, **mgWriteTexture**.

COPYRIGHT

mgIsTextureDefault

NAME

mgIsTextureDefault - finds out if the default pattern has been assigned to a texture palette entry.

SYNOPSIS

#include lapitexture.h

mgbool mgIsTextureDefault (mgrec* db, int index);

DESCRIPTION

If the load of a texture palette entry fails (due to file open failure, etc.), a default pattern (a black X on a white background) is substituted. **mgIsTextureDefault** returns **mgTRUE** if the specified texture palette entry has the default pattern.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry in question.

RETURNS

mgbool	returns **mgTRUE** if the texture has the default texels; returns **mgFALSE** and sets an error on failure.

ACCESS LEVEL

Level 1

SEE ALSO

mgIsTextureInPalette.

COPYRIGHT

mgIsTextureInPalette

NAME

mgIsTextureInPalette - indicates whether or not a texture is in a database's texture palette.

SYNOPSIS

#include lapitexture.h

mgbool mgIsTextureInPalette (mgrec* db, char *name);

DESCRIPTION

mgIsTextureInPalette returns **mgTRUE** if there is an entry in the given database's texture palette with the given *name*.

ARGUMENTS

db	a database node.
name	the name of the texture in question.

RETURNS

mgbool	returns **mgTRUE** if the texture is in the database's texture palette; returns **mgFALSE** and sets an error on failure.

ACCESS LEVEL

Level 1

SEE ALSO

mgIsTextureDefault, **mgGetTextureCount**.

COPYRIGHT

mgReadImage

NAME

mgReadImage - reads an image from disk.

SYNOPSIS

#include lapitexture.h

int mgReadImage (char *filename, unsigned char **pixels, int *type, int *width, int *height);

DESCRIPTION

Given an image file name, **mgReadImage** reads an image from disk into a pre-allocated array of pixels. The image *type*, *height*, and *width* are returned through arguments. A texture read status code is returned.

ARGUMENTS

filename	the name of the image file on disk.
pixels	a pointer to the image pixels.
type	a pointer to the image type
width	the width of the image.
height	the height of the image.

RETURNS

int	returns a status code:
	TXTIO_NO_ERROR = 0 = Success.
	TXTIO_MALLOC_ERR = -1 = Memory Allocation Failure.
	TXTIO_OPEN_ERR = -2 = Can’t Open File or File Not Found.
	TXTIO_READ_ERR = -3 = Read Error.
	TXTIO_SEEK_ERR = -5 = File Seek Error.
	TXTIO_BAD_FILE_TYPE = -6 = Unknown File Type.
	TXTIO_IMAGE_TOO_BIG = -7 = Image Too Large.
	TXTIO_BAD_DATA = -8 = Bad Data in Image File Header.
pixels	returns the image pixels.
type	returns the image type
	0 = Unused.
	1 = Unused.
	2 = Intensity (1 channel).
	3 = Intensity-Alpha (2channels).
	4 = RGB (3 channels).
	5 = RGBA (4 channels).
width	returns the width of the image.
height	returns the height of the image.

ACCESS LEVEL

Level 1

SEE ALSO

mgWriteImage, **mgReadImageHeader**, **mgReadTexture**, **mgInsertTexture**.

COPYRIGHT

mgReadImageAttributes

NAME

mgReadImageAttributes - reads an image's attributes file from disk.

SYNOPSIS

#include lapitexture.h

mgbool mgReadImageAttributes (char* filename, mgrec* rec);

DESCRIPTION

mgReadImageAttributes reads an image's attributes file from disk and fills an attributes record. The **imageFileName** passed to this function is the name of the image file. The attributes file name is *filename* appended with ".attr".

ARGUMENTS

filename	the name of the image on disk.
rec	a pointer to the attributes record to fill.

RETURNS

mgbool	returns **mgTRUE** if the file is read successfully; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 1

SEE ALSO

mgReadImage, **mgWriteImageAttributes**.

COPYRIGHT

mgReadImageHeader

NAME

mgReadImageHeader - reads the header information from an image file.

SYNOPSIS

#include lapitexture.h

int mgReadImageHeader (char *filename, int *type, int *width, int *height);

DESCRIPTION

mgReadImageHeader reads the header information of an image from the specified *filename*. The header information (type, width, and height) are returned, as well as status.

ARGUMENTS

filename	the name of the image file on disk.
type	a pointer to the image type
width	a pointer to the width return variable.
height	a pointer to the height return variable.

RETURNS

int	returns a status code:
	TXTIO_NO_ERROR = 0 = Success.
	TXTIO_MALLOC_ERR = -1 = Memory Allocation Failure.
	TXTIO_OPEN_ERR = -2 = Can’t Open File or File Not Found.
	TXTIO_READ_ERR = -3 = Read Error.
	TXTIO_SEEK_ERR = -5 = File Seek Error.
	TXTIO_BAD_FILE_TYPE = -6 = Unknown File Type.
	TXTIO_BAD_DATA = -8 = Bad Data in Image File Header.
type	returns the image type
	0 = Unused.
	1 = Unused.
	2 = Intensity (1 channel).
	3 = Intensity-Alpha (2channels).
	4 = RGB (3 channels).
	5 = RGBA (4 channels).
width	returns the width of the image.
height	returns the height of the image.

ACCESS LEVEL

Level 1

SEE ALSO

mgReadImage, **mgWriteImage**.

COPYRIGHT

mgReadTexture

NAME

mgReadTexture - loads a texture palette entry from a given file.

SYNOPSIS

#include lapitexture.h

mgbool mgReadTexture (mgrec* db, char *filename, int index, int x, int y);

DESCRIPTION

mgReadTexture loads a texture (image and attributes) into a database's texture palette at the specified index and location. The location (x, y) is the location of the lower left corner of the texture (as displayed in MultiGen) relative to the lower left corner of the palette. The attributes are read from the file specified by the texture file name appended with ".attr". If there is no attributes file, default attributes are assigned.

ARGUMENTS

db	a database node.
filename	the name of the texture file on disk.
index	the index to assign to the texture palette entry.
x, y	the position of the texture in the palette.

RETURNS

mgbool	returns **mgTRUE** if the image is read successfully; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 1

SEE ALSO

mgReadTextureAndAlpha, **mgInsertTexture**, **mgInsertTextureAndAlpha**, **mgWriteTexture**, **mgReadImage**.

COPYRIGHT

mgReadTextureAndAlpha
mgInsertTextureAndAlpha

NAME

mgReadTextureAndAlpha - loads a texture and alpha mask as a single texture palette entry. User defines the texture palette position, *x,y* and texture palette *index*.

mgInsertTextureAndAlpha - loads a texture and alpha mask as a single texture palette entry at a specified index. Texture palette position is assigned automatically.

SYNOPSIS

#include lapitexture.h

mgbool mgReadTextureAndAlpha (mgrec* db, char *textureFileName, char *alphaFileName, char *mergeTextureName, int index, int x, int y);

mgbool mgInsertTextureAndAlpha (mgrec* db, char *textureFileName, char *alphaFileName, char *mergeTextureName);

DESCRIPTION

mgInsertTextureAndAlpha loads a texture and alpha mask (stored on disk as two separate image files) into a database's texture palette at the specified index. The texture and alpha are merged into a single texture and assigned the given index. The merged texture is identified in the palette by *mergeTextureName*. This function automatically positions the texture so no other textures are covered.

mgReadTextureAndAlpha acts in the same way, but adds a user-defined position, *x,y* and *index*. The location (x, y) is the location of the lower left corner of the texture in the palette (as displayed in MultiGen) relative to the lower left corner of the palette.

ARGUMENTS

db a database node.

textureFileName

the name of the texture file on disk.

alphaFileName

the name of the alpha mask file to load.

mergeTextureName

the name to associate with the merged texture.

index the index to assign to the texture palette entry (**mgReadTextureAndAlpha** only).

x, y the position of the entry in the palette (**mgReadTextureAndAlpha** only).

RETURNS

mgbool returns **mgTRUE** if the image is read successfully; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 1

SEE ALSO

mgReadTexture, **mgInsertTexture**, **mgWriteTexture**.

COPYRIGHT

mgReadTexturePalette

NAME

mgReadTexturePalette - reads a texture palette file from disk.

SYNOPSIS

#include lapitexture.h

int mgReadTexturePalette (mgrec* db, char *filename);

DESCRIPTION

Given a database node, *db*, **mgReadTexturePalette** loads the textures listed in a texture palette file, *filename*, into the database's texture palette. If the database's existing texture palette contains entries with the same indices as the ones in the texture palette file, the new entries overwrite the ones currently in the texture palette. The return value is the number of entries loaded.

ARGUMENTS

db	a database node.
filename	the texture palette file name.

RETURNS

int	returns the number of textures loaded if successful; otherwise returns **-1.**

ACCESS LEVEL

Level 1

SEE ALSO

mgWriteTexturePalette.

COPYRIGHT

mgReplaceTexture

NAME

mgReplaceTexture - replaces a texture palette entry with a texture from a given file.

SYNOPSIS

#include lapitexture.h

mgbool mgReplaceTexture (mgrec *db, int index, char *filename);

DESCRIPTION

mgReplaceTexture replaces a texture palette entry specified by *index* with an image and attributes from files specified by *filename* (the attributes file is filename appended with ".attr"). If there is no attributes file, default attributes are used.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.
filename	the texture filename.

RETURNS

mgbool	returns **mgTRUE** if the entry was replaced successfully; otherwise **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgReadTextureAndAlpha, **mgReadTexture**.

COPYRIGHT

mgSetTextureAttributes

NAME

mgSetTextureAttributes - sets the attributes of a texture palette entry.

SYNOPSIS

#include lapitexture.h

void mgSetTextureAttributes (mgrec *db, int index, mgrec *att_rec);

DESCRIPTION

Given a database node, *db*, and a texture palette entry specified by *index*, **mgSetTextureAttributes** sets the attributes record of the texture palette entry to the given attributes record, *att_rec*.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.
att_rec	a pointer to the attributes record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgGetTextureAttributes.

COPYRIGHT

mgSetTextureName

NAME

mgSetTextureName - sets the name of a texture palette entry.

SYNOPSIS

#include lapitexture.h

mgbool mgSetTextureName (mgrec *db, int index, char *textureName);

DESCRIPTION

mgSetTextureName sets the name of the texture palette entry, specified by *index*, to *textureName*.

ARGUMENTS

db a database node.

index the index of the texture palette entry.

textureName
the new name of the texture.

RETURNS

mgbool returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgGetTextureName.

COPYRIGHT

mgSetTexturePosition

NAME

mgSetTexturePosition - sets the position of a texture palette entry.

SYNOPSIS

#include lapitexture.h

mgbool mgSetTexturePosition (mgrec *db, int index, int x, int y);

DESCRIPTION

mgSetTexturePosition sets the position of the lower left corner of the specified texture palette entry in a database's palette (as displayed in MultiGen). The position is relative to the lower left corner of the palette.

ARGUMENTS

db_rec	a database node.
index	the index of the texture palette entry.
x, y	the position of the texture.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgGetTexturePosition.

COPYRIGHT

mgSetTextureTexels

NAME

mgSetTextureTexels - sets the texels of a texture palette entry.

SYNOPSIS

#include lapitexture.h

mgbool mgSetTextureTexels (mgrec *db, int index, unsigned char *texels);

DESCRIPTION

mgSetTextureTexels sets the texels for the texture palette entry specified by *index*. The width, height, and type of the new image must match the width, height, and type of the original image.

ARGUMENTS

db	a database node.
index	the index of the texture palette entry.
texels	a pointer to the new texels.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgGetTextureTexels.

COPYRIGHT

mgWriteImage

NAME

mgWriteImage - writes an image to disk.

SYNOPSIS

#include lapitexture.h

int mgWriteImage (char *filename, unsigned char *pixels, int type, int width, int height, int compress);

DESCRIPTION

mgWriteImage writes an image to disk in the SGI RGB format using the given *filename*. If the compress flag is **mgTRUE**, the image is compressed using SGI's run-length encoding scheme.

ARGUMENTS

filename	the name of the image file on disk.
pixels	the image pixels.
type	the image type.
	0 = Unused.
	1 = Unused.
	2 = Intensity (1 channel).
	3 = Intensity-Alpha (2channels).
	4 = RGB (3 channels).
	5 = RGBA (4 channels).
width	the image width.
height	the image height.
compress	a flag specifying whether or not to compress the image.

RETURNS

int	returns a status code:
	TXTIO_NO_ERROR = 0 = Success.
	TXTIO_MALLOC_ERR = -1 = Memory Allocation Failure.
	TXTIO_OPEN_ERR = -2 = Can't Open File, File Not Found, or no Write Permission.
	TXTIO_WRITE_ERR = -4 = Write Error.
	TXTIO_SEEK_ERR = -5 = File Seek Error.

ACCESS LEVEL

Level 2

SEE ALSO

mgReadImage, **mgReadImageHeader**, **mgWriteTexture**.

COPYRIGHT

mgWriteImageAttributes

NAME

mgWriteImageAttributes - writes image attributes to disk.

SYNOPSIS

#include lapitextures.h

mgbool mgWriteImageAttribute (char *filename, mgrec *rec);

DESCRIPTION

mgWriteImageAttribute writes an image's attributes file to disk using the given *filename* appended with ".attr".

ARGUMENTS

filename	the image attributes file name, without the extension ".attr".
rec	a pointer to the image attributes record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgReadImageAttributes, **mgWriteImage**.

COPYRIGHT

mgWriteTexture

NAME

mgWriteTexture - writes a texture palette entry to disk.

SYNOPSIS

#include lapitexture.h

mgbool mgWriteTexture (mgrec* db, int index, char* filename);

DESCRIPTION

mgWriteTexture writes the specified texture image to disk, in SGI's RGB format, using the given *filename*, and writes the attributes to disk using the given *filename* appended with ".attr".

ARGUMENTS

db	a database node.
index	the texture palette index.
filename	the name of the texture file on disk.

RETURNS

mgbool	returns **mgTRUE** on success; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgWriteImage, **mgReadTexture**.

COPYRIGHT

mgWriteTexturePalette

NAME

mgWriteTexturePalette - writes the contents of a database's texture palette to disk.

SYNOPSIS

#include lapictexture.h

mgbool mgWriteTexturePalette (mgrec* db, char *paletteFileName);

DESCRIPTION

mgWriteTexturePalette writes an ascii file listing the textures in the specified database's palette.

ARGUMENTS

db	a database node.
name	the texture palette file name.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgReadTexturePalette.

COPYRIGHT

mgGetFirstTextureMappingInPalette
mgGetNextTextureMappingInPalette

NAME

mgGetFirstTextureMappingInPalette - gets the first texture mapping in a database's palette.

mgGetNextTextureMappingInPalette - gets successive texture mappings in a database's palette.

SYNOPSIS

#include mgapitxtrmap.h

mgbool mgGetFirstTextureMappingInPalette (mgrec* db, int *index, char *name);

mgbool mgGetNextTextureMappingInPalette (mgrec* db, int *index, char *name);

DESCRIPTION

mgGetFirstTextureMappingInPalette gets the index and name of the first texture mapping in a database's palette. **mgGetNextTextureMappingInPalette** gets the indices and names of the remainder of the texture mappings in the palette after **mgGetFirstTextureMappingInPalette** is used.

EXAMPLE

```
int index;
mgrec *db;
char name [ 200 ];
if ( mgGetFirstTextureMappingInPalette ( db, &index, name ) ) {
  while ( mgGetNextTextureMappingInPalette ( db, &index, name ) ) {
  }
}
```

ARGUMENTS

db	a database record.
index	the address of the index variable.
name	the name of the texture.

RETURNS

mgbool	returns **mgTRUE** if a texture is found; otherwise returns **mgFALSE**. **mgFALSE** should be used to flag the end of the traversal.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureMappingName

COPYRIGHT

mgGetTextureMappingMatrix

NAME

mgGetTextureMappingMatrix - gets the mapping defined by *index*, and puts the mapping's matrix into *matrix*.

SYNOPSIS

#include mgapitxtrmap.h

mgbool mgGetTextureMappingMatrix (mgrec* db, int *index, double matrix[4][4]);

DESCRIPTION

mgGetTextureMappingMatrix gets the mapping defined by *index*, and puts the mapping's matrix into *matrix*. The result is suitable for use with OpenGL's glTexGen function linear mappings. The matrix is only valid for 3 point put and 4 point put mappings.

gets the mapping defined by *index*, and puts the mapping's matrix into *matrix*.

EXAMPLE

```
int index;
mgrec *db;
double matrix[4][4];
if ((mgGetTextureMappingType ( db, index ) == 1 ) || (mgGetTextureMappingType ( db, index ) == 2 ))
{
        if( mgGetTextureMappingMatrix ( db, index, matrix ) )
        {
        /* Do Stuff */
        }
}
```

ARGUMENTS

db	a database record.
index	the mapping's index.
matrix	a matrix to fill.

RETURNS

mgbool	returns **mgTRUE** if a valid matrix is found; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureMappingName

COPYRIGHT

mgGetTextureMappingName

NAME

mgGetTextureMappingName - gets the name of a texture mapping.

SYNOPSIS

#include mgapitxtrmap.h

char * mgGetTextureMappingName (mgrec* db, int *index, char *textureName ;

DESCRIPTION

mgGetTextureMappingName returns the name of a texture mapping.

EXAMPLE

int index;

mgrec *db;

char* name;

name = **mgGetFirstTextureInPalette** (db, index,))

ARGUMENTS

db	a database record.
index	the address of the index variable.

RETURNS

char *	returns a pointer to the mapping name if a mapping is found and the mapping has a name;; otherwise returns **mgNULL.**

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureMappingType

COPYRIGHT

mgGetTextureMappingType

NAME

mgGetTextureMappingType - gets the mapping type of a texture.

SYNOPSIS

#include mgapitxtrmap.h

int mgGetTextureMappingType (mgrec* db, int *index);

DESCRIPTION

mgGetTextureMappingType gets the texture mapping type. The returned types are:

0 = Error
1 = 3 point put
2 = 4 point put
3 = Not implemented
4 = Spherical project
5 = Radial project
6 = Not implemented

EXAMPLE

```
int index;
mgrec *db;
switch ( mgGetTextureMappingType ( db, index )
{
        case 1:
        /* 3 point put */
        break;
        case 2:
        /* 4 point put */
        break;
        case 4:
        /* Shperical project */
        break;
        case 5:
        /* Radial project */
        break;
        default:
        /* error */
        break;
}
```

ARGUMENTS

db	a database record.
index	the address of the index variable.

RETURNS

int returns the mapping type if a texture is found; otherwise returns 0.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetTextureMappingName

COPYRIGHT

mgIsTextureMappingInPalette

NAME

mgIsTextureMappingInPalette - determines if a texture mapping is in a database's palette.

SYNOPSIS

#include mgapitxtrmap.h

mgbool mgIsTextureMappingInPalette (mgrec* db, int index);

DESCRIPTION

mgIsTextureMappingInPalette reports when a mapping with the given index is in the palette of a database, and when the given index is not found**.**

EXAMPLE

```
int index;
mgrec *db;
if ( mgIsTextureMappingInPalette ( db, index ) )
{
    /* Do Stuff */
}
```

ARGUMENTS

db	a database record.
index	the index of the texture mapping.

RETURNS

mgbool	returns **mgTRUE** if a texture mapping is found; otherwise returns **mgFALSE.** On error, returns **mgFALSE** and sets an api error.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetFirstTextureMappingInPalette, **mgGetNextTextureMappingInPalette**

COPYRIGHT

mgDeleteComment

NAME

mgDeleteComment - deletes the comment string of a node record.

SYNOPSIS

#include lapiattr.h

mgbool mgDeleteComment (mgrec* rec);

DESCRIPTION

mgDeleteComment deallocates the memory used (if any) by the comment string of the node record *rec*. Note that vertex node records do not have comment strings.

ARGUMENTS

rec	points to the node record whose comment string is to be deleted.

RETURNS

mgbool	returns **mgTRUE** on success; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgGetComment, **mgSetComment**

COPYRIGHT

mgGetAttList

NAME

mgGetAttList - gets the values of record attributes using the **varargs** argument style

SYNOPSIS

#include lapiattr.h

int mgGetAttList (mgrec* rec, <att_code>, <att_var>, ..., mgNULL);

DESCRIPTION

Given a record, *rec*, and a list of attribute code/variable pairs, **mgGetAttList** gets the values of *rec*'s attributes named by the attribute codes and stores them in the corresponding attribute variables. It is the caller's responsibility to ensure that sufficient storage is allocated for each of the attribute values. *rec* cannot be a value record.

Variable-length argument lists cannot be type-checked by the compiler, so using **varargs** functions can be a source of bugs. Be sure to end all of your argument lists with **mgNULL** and be sure that the type of each argument is as expected.

ARGUMENTS

rec	points to the record from which to get values.
att_code	the attribute typecode.
att_var	the pointer to the returned attribute value.

RETURNS

int	returns the number of attribute code/variable pairs evaluated.
att_var	returns the returned attribute value.

EXAMPLE

```
mgrec *poly_rec;
short poly_texture;
short poly_material;


mgGetAttList (poly_rec,
              fltPolyMaterial, &poly_material,
              fltPolyTexture1, &poly_texture,
              mgNULL);
```

ACCESS LEVEL

Level 1

SEE ALSO

mgSetAttList

COPYRIGHT

mgGetAttBuf

NAME

mgGetAttBuf - gets consecutive values from an attribute record and store them in a data buffer.

SYNOPSIS

#include lapiattr.h

mgbool mgGetAttBuf (mgrec* rec, mgcode fcode, void *buf);

DESCRIPTION

Given a record, *rec*, **mgGetAttBuf** gets the values of the attribute *fcode* from *rec* and stores those values in the data buffer *buf*. It is the caller's responsibility to ensure that sufficient storage is allocated.

This routine is used for setting all the attribute values of a record, with the condition that the attributes are consecutive as defined by the data dictionary. Note: simple records such as **fltIcoord** and **fltMatrix** are guaranteed to be stored consecutively; more complex records are not. Exercise caution when using this routine.

ARGUMENTS

rec	points to the record from which to get values.
fcode	the attribute code.
buf	points to the storage area for the attribute values.

RETURN

mgbool	returns **mgTRUE** if data is stored in the buffer; otherwise returns **mgFALSE**.
buf	the attribute values, stored in a contiguous buffer.

EXAMPLE

```
mgrec *poly_rec;
double mtx[16];


mgGetAttBuf (poly_rec, fltMatrix, mtx);
```

ACCESS LEVEL

Level 1

SEE ALSO

mgSetAttBuf

COPYRIGHT

mgGetAttRec

NAME

mgGetAttRec - gets a pointer to one of a record's attribute records.

SYNOPSIS

#include lapiattr.h

mgbool mgGetAttRec (mgrec* rec, mgcode fcode, mgrec *rec_out);

DESCRIPTION

Given a record, *rec*, **mgGetAttRec** gets a pointer to the attribute record *fcode* from *rec* and stores it in *rec_out*.

mgGetAttRec is used for retrieving an attribute record as opposed to an attribute value. Use **mgGetAttList** to retrieve an attribute value.

ARGUMENTS

rec	points to the record from which to get the attribute record.
fcode	the attribute record code.
rec_out	points to the attribute record.

RETURNS

mgbool	returns **mgTRUE** if returning an attribute record pointer; otherwise returns **mgFALSE**.
rec_out	returns the attribute record pointer.

EXAMPLE

```
mgrec *group_rec;
mgrec *att_rec;

int bound_type;


mgGetAttRec (group_rec, fltBoundingBox, att_rec);

mgGetAttList ( att_rec, fltBoundingType, bound_type, mgNULL );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgSetAttRec, **mgGetAttList**, **mgGetAttBuf**, **mgGetAttXmBuf**

COPYRIGHT

mgSetAttBuf

NAME

mgSetAttBuf - using a data buffer, sets the values of consecutive attributes.

SYNOPSIS

#include lapiattr.h

mgbool mgSetAttBuf (mgrec* rec, mgcode fcode, void *buf);

DESCRIPTION

Given a record, *rec*, **mgSetAttBuf** sets the values of the attribute named by *fcode* from *buf*. This routine is used to set all the attribute values of a record with the condition that the attributes are consecutive as defined by the data dictionary. Note: simple records such as fltIcoord and fltMatrix values are guaranteed to be stored consecutively; more complex records are not. Exercise caution when using this routine.

ARGUMENTS

rec	points to the record in which to set values.
fcode	the code of the attribute to be set from *buf*.
buf	the data buffer from which to assign attribute values.

RETURNS

mgbool	returns **mgTRUE** if the attribute values are assigned; otherwise returns **mgFALSE**.

EXAMPLE

```
mgrec *vtx_rec;
double icoord1 = { 1., 0., 0. };

mgSetAttBuf (vtx_rec, fltIcoord, icoord1 );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgGetAttBuf

COPYRIGHT

mgGetAttXmBuf

NAME

mgGetAttXmBuf - gets the current values of transformation record attributes in a data buffer

SYNOPSIS

#include lapiattr.h

mgbool mgGetAttXmBuf (mgrec* rec, mgcode fcode, void *buf);

DESCRIPTION

Given a transformation record, *rec*, **mgGetAttXmBuf** gets the data of the attribute *fcode* from *rec* and stores those values in data buffer *buf*. It is the caller's responsibility to ensure that sufficient storage is allocated.

mgGetAttXmBuf is used for retrieving all the data from a transformation record aggregate attribute.

ARGUMENTS

rec	points to the transformation record from which to get values.
fcode	the attribute code.
buf	points to the storage area for the attribute values.

RETURNS

mgbool	returns **mgTRUE** if data is stored in the buffer; otherwise returns **mgFALSE**.
buf	returns the transformation record attribute values, stored in a contiguous buffer.

EXAMPLE

```
mgrec *poly_rec;
double mtx[16];

mgGetAttBuf (poly_rec, fltXmTranslate, mtx);
```

ACCESS LEVEL

Level 1

SEE ALSO

mgGetAttBuf

COPYRIGHT

mgGetComment

NAME

mgGetComment - gets a copy of the comment string of a node record.

SYNOPSIS

#include lapiattr.h

char* mgGetComment (mgrec* rec);

DESCRIPTION

Given a node record, *rec*, **mgGetComment** returns a copy of the string that is the node's comment string. The storage for the returned string is dynamically allocated by **mgGetComment**. Use **mgSetComment** or **mgDeleteComment** to change or delete the record's actual comment string). Note that vertex node records do not have comment strings.

Note: the user is responsible for freeing the dynamically allocated memory with **mgFree**.

ARGUMENTS

rec	points to the node record whose comment string is desired.

RETURNS

char*	returns a copy of the node's comment string if successful; otherwise returns **mgNULL**.

ACCESS LEVEL

Level 1

SEE ALSO

mgSetComment, **mgDeleteComment**

COPYRIGHT

mgGetName

NAME

mgGetName - gets a copy of the name of a node record.

SYNOPSIS

#include lapiattr.h

char* mgGetName (mgrec* rec);

DESCRIPTION

Given a node record, *rec*, **mgGetName** returns the string that is the node's ID. The storage for the string is dynamically allocated by **mgGetName**. Use **mgSetName** to change the name of a record. The node record in question must have a name attribute.

Note: the user is responsible for deallocating the dynamically allocated memory with **mgFree**.

ARGUMENTS

rec	the node record.

RETURNS

char*	returns a copy of node's name string; otherwise returns **mgNULL**.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetRecByName, **mgSetName**

COPYRIGHT

mgSetAttList

NAME

mgSetAttList - sets the values of record attributes using varargs argument style

SYNOPSIS

#include lapiattr.h

int mgSetAttList (mgrec* rec, <att_code>, <att_val>,..., mgNULL);

DESCRIPTION

Given a record, *rec*, and a list of attribute code/value pairs, *<att_code>*, *<att_val>*, **mgSetAttList** sets the values of *rec*'s attributes named by the attribute code to corresponding attribute values. *rec* cannot be a value record.

Variable-length argument lists cannot be type-checked by the compiler, so using varargs functions can be a source of bugs. Be sure to end all of your argument lists with **mgNULL** and be sure that the type of each argument is as expected.

ARGUMENTS

rec	points to the record in which to set attribute values.
att_code	the attribute typecode.
att_val	the attribute value.

RETURNS

int	returns the number of attribute code/address pairs evaluated.

EXAMPLE

```
mgrec *poly_rec;
short poly_texture = 1;
short poly_material = 1;

mgSetAttList (poly_rec,
              fltPolyMaterial, poly_material,
              fltPolyTexture1, poly_texture,
              mgNULL);
```

ACCESS LEVEL

Level 1

SEE ALSO

mgGetAttList, **mgGetAttBuf**, **mgSetAttBuf**

COPYRIGHT

mgSetColorRGBA

NAME

mgSetColorRGBA - sets the attribute values in a fltColorRGBA record.

SYNOPSIS

#include mglapiattr.h

mgbool mgSetColorRGBA (mgrec* rec, mgcode name, float r, float g, float b, float a);

DESCRIPTION

Given a record and the name of one of its **fltColorRGBA** attribute records, **mgSetColorRGBA** sets the r, g, b, and a attribute values in the **fltColorRGBA** attribute record.

ARGUMENTS

rec	points to the record whose **fltColorRGBA** record is to be set.
name	the name of the **fltColorRGBA** attribute record of rec.
r	the r value to set in the **fltColorRGBA** attribute record.
g	the g value to set in the **fltColorRGBA** attribute record.
b	the b value to set in the **fltColorRGBA** attribute record.
a	the a value to set in the **fltColorRGBA** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetPackedColor, **mgSetNormColor**

COPYRIGHT

mgSetComment

NAME

mgSetComment - assigns a comment string to a node record.

SYNOPSIS

#include lapiattr.h

mgbool mgSetComment (mgrec* rec, char *comment);

DESCRIPTION

mgSetComment allocates space and copies *comment* into the comment string of the node record *rec*. Pre-existing comments are deleted before the new comment space is allocated. Note that vertex node records do not have comment strings.

ARGUMENTS

rec	points to the node record whose comment string is to be set.
comment	the new comment string for the node record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgGetComment, **mgDeleteComment**.

COPYRIGHT

mgSetFcoord

NAME

mgSetFcoord - sets all three attribute values in an Fcoord record.

SYNOPSIS

#include lapiattr.h

mgbool mgSetFcoord (mgrec* rec, mgcode name, float x, float y, float z);

DESCRIPTION

Given a record and the name of one of its **fltFcoord** attribute records, **mgSetFcoord** sets the x, y, and z attribute values in the **fltFcoord** attribute record.

ARGUMENTS

rec	points to the record whose **fltFcoord** record is to be set.
name	the name of the **fltFcoord** attribute record of rec.
x	the x value to set in the **fltFcoord** attribute record.
y	the y value to set in the **fltFcoord** attribute record.
z	the z value to set in the **fltFcoord** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetICoord

COPYRIGHT

mgSetIcoord

NAME

mgSetIcoord - sets all three attribute values in an Icoord record.

SYNOPSIS

#include lapiattr.h

mgbool mgSetIcoord (mgrec* rec, mgcode name, double x, double y, double z);

DESCRIPTION

Given a record and the name of one of its **fltIcoord** attribute records, **mgSetIcoord** sets the x, y, and z attribute values in the **fltIcoord** attribute record.

ARGUMENTS

rec	points to the record whose **fltIcoord** record is to be set.
name	the name of the **fltIcoord** attribute record of *rec*.
x	the x value to set in the **fltIcoord** attribute record.
y	the y value to set in the **fltIcoord** attribute record.
z	the z value to set in the **fltIcoord** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetFCoord

COPYRIGHT

mgSetName

NAME

mgSetName - assigns a name to a node record.

SYNOPSIS

#include lapiattr.h

mgbool mgSetName (mgrec* rec, char *name);

DESCRIPTION

mgSetName copies *name* into the name field of the node record *rec.* The node record must have a name attribute as defined by the OpenFlight data dictionary.

ARGUMENTS

rec	a node record.
name	the new name for the node record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

COPYRIGHT

mgSetNormColor

NAME

mgSetNormColor - sets attribute values in a **fltNormColor** record.

SYNOPSIS

#include lapiattr.h

mgbool mgSetNormColor (mgrec* rec, mgcode name, float r, float g, float b);

DESCRIPTION

Given a record and the name of one of its **fltNormColor** attribute records, **mgSetNormColor** sets the r, g, and b attribute values in the **fltNormColor** attribute record.

ARGUMENTS

rec	points to the record whose **fltNormColor** record is to be set.
name	the name of the **fltNormColor** attribute record of rec.
r	the red value to set in the **fltNormColor** attribute record.
g	the green value to set in the **fltNormColor** attribute record.
b	the blue value to set in the **fltNormColor** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetColorRGBA, **mgSetPackedColor**.

COPYRIGHT

mgSetPackedColor

NAME

mgSetPackedColor - sets the attribute values in an **fltPackedColor** record.

SYNOPSIS

#include mgapiattr.h

mgbool mgSetPackedColor (mgrec* rec, mgcode name, unsigned char r, unsigned char g, unsigned char b);

DESCRIPTION

Given a record and the name of one of its **fltPackedColor** attribute records, **mgSetPackedColor** sets the r, g, b, and a attribute values in the **fltPackedColor** attribute record.

ARGUMENTS

rec	points to the record whose **fltPackedColor** record is to be set.
name	the name of the **fltPackedColor** attribute record of rec.
r	the r value to set in the **fltPackedColor** attribute record.
g	the g value to set in the **fltPackedColor** attribute record.
b	the b value to set in the **fltPackedColor** attribute record.
a	the a value to set in the **fltPackedColor** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetColorRGBA, **mgSetNormColor**

COPYRIGHT

mgSetIPoint

NAME

mgSetIPoint - sets all three attribute values in a **fltIPoint** record.

SYNOPSIS

#include mgapiattr.h

mgbool mgSetIPoint (mgrec* rec, mgcode name, int x, int y);

DESCRIPTION

Given a record and the name of one of its **fltIPoint** attribute records, **mgSetIPoint** sets the x and y attribute values in the **fltIPoint** attribute record.

ARGUMENTS

rec	points to the record whose **fltIPoint** record is to be set.
name	the name of the **fltIPoint** attribute record of rec.
x	the x value to set in the **fltIPoint** attribute record.
y	the y value to set in the **fltIPoint** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetIcoord

COPYRIGHT

mgSetVector

NAME

mgSetVector - sets the attribute values in an **fltVector** record.

SYNOPSIS

#include mgapiattr.h

mgbool mgSetVector (mgrec* rec, mgcode name, float i, float j, float k);

DESCRIPTION

Given a record and the name of one of its **fltVector** attribute records, **mgSetVector** sets the x, y, and z attribute values in the **fltVector** attribute record.

ARGUMENTS

rec	points to the record whose **fltVector** record is to be set.
name	the name of the **fltVector** attribute record of rec.
i	the i value to set in the **fltVector** attribute record.
j	the j value to set in the **fltVector** attribute record.
k	the k value to set in the **fltVector** attribute record.

RETURNS

mgbool	returns **mgTRUE** if successful; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgSetICoord.

COPYRIGHT

mgDeleteColorName

NAME

mgDeleteColorName - deletes a name from a color palette entry's color name list.

SYNOPSIS

#include lapicolor.h

mgbool mgDeleteColorName (mgrec* db, int index, char *name);

DESCRIPTION

Given a database node, *db*, and a color palette index, *index*, **mgDeleteColorName** deletes the color name *name* from the color name list of *index*. An index of **-1** means "search the entire color palette."

ARGUMENTS

db	a database node.
index	the color palette index, or **-1** for the entire palette.
name	the color name to delete from the entry's name list.

RETURNS

mgbool	returns **mgTRUE** if the name was successfully deleted from the name list; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgNewColorName, **mgSetCurrentColorName**.

COPYRIGHT

mgGetColorIndexByName

NAME

mgGetColorIndexByName - finds the index of a color palette entry from its name.

SYNOPSIS

#include lapicolor.h

mgbool mgGetColorIndexByName (mgrec* db, char *name, int *index);

DESCRIPTION

Given a database node, *db*, and a color name, *name*, **mgGetColorIndexByName** sets *index* to the index of the color palette entry that has *name* in its name list.

ARGUMENTS

db	a database node.
name	the color name
index	pointer for the returned index.

RETURNS

mgbool	returns **mgTRUE** if the named color was found; otherwise returns **mgFALSE.**
index	returns the index of the named color.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetCurrentColorName, **mgGetNextColorName**.

COPYRIGHT

mgGetCurrentColorName

NAME

mgGetCurrentColorName - get a copy of the current name of a particular color palette entry.

SYNOPSIS

#include lapicolor.h

char *mgGetCurrentColorName (mgrec* db, int index);

DESCRIPTION

Each color palette entry can have a list of names associated with it. One of these names is always the current name for that entry. Given a database node, *db*, and an *index* into the color palette, **mgGetCurrentColorName** returns the current name of the given color palette entry. The storage for the name is dynamically allocated by **mgGetCurrentColorName**.

Note: The user is responsible for deallocating the dynamically allocated memory with **mgFree**.

ARGUMENTS

db	a database node.
index	the color palette index.

RETURNS

char*	returns a copy of the current name for the color palette entry; otherwise returns **mgNULL**.

ACCESS LEVEL

Level 1

SEE ALSO

mgSetCurrentColorName, **mgGetNextColorName**, **mgGetColorIndexByName**.

COPYRIGHT

mgGetNextColorName

NAME

mgGetNextColorName - gets the name of the next color for a particular color index.

SYNOPSIS

#include lapicolor.h

char *mgGetNextColorName (mgrec* db, int index, void **ptr)

DESCRIPTION

Each color palette entry can have a list of names associated with it. Given a database node, *db*, an *index* into the color palette, and the name list, *ptr*, **mgGetNextColorName** returns the next name in *index*'s color name list. If *ptr* is **mgNULL,** then it will be set to hold an opaque pointer to the name list, and the first name in the list will be returned. Subsequent calls to **mgGetNextColorName** for the same *index* can then be made using *ptr*. Storage for the name is dynamically allocated by **mgGetNextColorName**.

Note: The user is responsible for deallocating the dynamically allocated memory with **mgFree.**

ARGUMENTS

db	is a database node.
index	the color palette index.
ptr	opaque pointer holding the name list (can be **mgNULL**).

RETURNS

char*	returns the next name in the color name list for the given index.
ptr	if passed in initially as **mgNULL,** returns an opaque pointer to the name list.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetCurrentColorName, **mgGetColorIndexByName**.

COPYRIGHT

mgIndex2RGB

NAME

mgIndex2RGB - returns red, green, and blue values for a given color palette index and intensity.

SYNOPSIS

#include lapicolor.h

void mgIndex2RGB (mgrec *db, unsigned int index, unsigned float intensity, short* r, short* g, short* b);

DESCRIPTION

Given a database node *db*, color palette *index* and *intensity*, **mgIndex2RGB** returns the red, green, and blue values associated with them in the database's color palette.

ARGUMENTS

db	the database node.
index	the index into the color palette
intensity	the intensity component of the color (0-127)
r	the pointer to the returned red value
g	the pointer to the returned green value
b	the pointer to the returned blue value

RETURNS

r	the returned red value, which ranges from 0 to 255
g	the returned green value, which ranges from 0 to 255
b	the returned blue value, which ranges from 0 to 255

ACCESS LEVEL

Level 1

SEE ALSO

mgRGB2Index.

COPYRIGHT

mgReadDefaultColorPalette

NAME

mgReadDefaultColorPalette - reads in a color file as the default color palette.

SYNOPSIS

#include lapicolor.h

mgbool mgReadDefaultColorPalette (char *filename);

DESCRIPTION

Given a color filename, reads the color file as the default color palette.

ARGUMENTS

filename the color file name.

RETURNS

mgbool returns **mgTRUE** if the file was read successfully, otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgReadColorPalette, **mgWriteDefaultColorPalette**.

COPYRIGHT

mgReadColorPalette

NAME

mgReadColorPalette - reads in a color file as a database's current color palette.

SYNOPSIS

#include lapicolor.h

mgbool mgReadColorPalette (mgrec *db_rec, char *filename);

DESCRIPTION

Given a database top node and color filename, reads the color file as the database's current color palette.

ARGUMENTS

db_rec a database node.

filename the color file name.

RETURNS

mgbool returns **mgTRUE** if the file was read successfully, otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgReadDefaultColorPalette, **mgWriteDefaultColorPalette**.

COPYRIGHT

mgWriteDefaultColorPalette

NAME

mgWriteDefaultColorPalette - writes the default color palette as a file on disk.

SYNOPSIS

#include lapicolor.h

mgbool mgWriteDefaultColorPalette (char *filename);

DESCRIPTION

Given a color filename, writes the default color palette to disk with this filename.

ARGUMENTS

filename is the file name for the default color palette.

RETURNS

mgbool returns **mgTRUE** if the file was written successfully; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgReadColorPalette, **mgReadDefaultColorPalette**, **mgWriteColorPalette**.

COPYRIGHT

mgWriteColorPalette

NAME

mgWriteColorPalette - writes a database's color palette as a file on disk.

SYNOPSIS

#include lapicolor.h

mgbool mgWriteColorPalette (mgrec *db_rec, char *filename);

DESCRIPTION

Given a database top node *db rec*, and color file name *filename*, writes the database's color file to disk.

ARGUMENTS

db_rec	a database top node.
filename	the color file name.

RETURNS

mgbool	returns **mgTRUE** if the file was written successfully; otherwise returns **mgFALSE**

ACCESS LEVEL

Level 2

SEE ALSO

mgReadColorPalette, **mgReadDefaultColorPalette**, **mgWriteDefaultColorPalette**.

COPYRIGHT

mgNewColorName

NAME

mgNewColorName - adds a name to a color palette entry's name list.

SYNOPSIS

#include lapicolor.h

mgbool mgNewColorName (mgrec* db, int index, char *name);

DESCRIPTION

Given a database node, *db*, and an index into the color palette, *index*, **mgNewColorName** adds the color name *name* to the color name list for *index*. It is assumed that *name* is not already in the color name list of any index, therefore no search is performed.

ARGUMENTS

db	a database node.
index	the color palette index.
name	the name to add to the name list of index.

RETURNS

mgbool	returns **mgTRUE** if the name was successfully added to the name table; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgDeleteColorName, **mgSetCurrentColorName**.

COPYRIGHT

mgRGB2Index

NAME

mgRGB2Index - finds the color palette entry and intensity which most closely matches RGB values.

SYNOPSIS

#include lapicolor.h

void mgRGB2Index (mgrec *db, short r, short g, short b, unsigned int *pal_index, float *intensity);

DESCRIPTION

Given a database node *db*, and red, green and blue values *r*, *g*, *b*, **mgRGB2Index** returns the index and intensity of the closest match in the database's color palette. The best match is determined by the least sum of squares method.

ARGUMENTS

db	the database node
r	the red value which ranges from 0 to 255
g	the green value which ranges from 0 to 255
b	the blue value which ranges from 0 to 255
pal_index	the pointer to the index of the best match in the color palette
intensity	the pointer to the intensity for the best match color index (0-127)

RETURNS

pal_index	returns the index of the best match in the database's color palette
intensity	returns the intensity for the best match (0-127)

ACCESS LEVEL

Level 1

SEE ALSO

mgIndex2RGB.

COPYRIGHT

mgSetColorIndex

NAME

mgSetColorIndex - sets a color palette entry to the given red, green, and blue values.

SYNOPSIS

#include lapicolor.h

mgbool mgSetColorIndex (mgrec *db, int index, short r, short g, short b);

DESCRIPTION

Given a database node *db*, a color palette index, *index*, and red, green and blue values, *r*, *g*, *b*, **mgSetColorIndex** sets the color of the given *index* in the database's color palette.

ARGUMENTS

db	the database node.
index	the color palette index to change
r	the desired red value which ranges from 0 to 255
g	the desired green value which ranges from 0 to 255
b	the desired blue value which ranges from 0 to 255

RETURNS

mgbool	is **mgTRUE** if *db* contains a valid database; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 1

SEE ALSO

mgRGB2Index, **mgGetColorIndexByName**.

COPYRIGHT

mgSetCurrentColorName

NAME

mgSetCurrentColorName - sets the current name of a particular color palette entry.

SYNOPSIS

#include lapicolor.h

mgbool mgSetCurrentColorName (mgrec* db, int index, char *name);

DESCRIPTION

Each color palette entry can have a list of names associated with it. One of these names is always the current name for that entry. Given a database node, *db*, and an index of a color palette entry, *index*, **mgSetCurrentColorName** sets the current name of the given color index to *name*. An index of **-1** causes the routine to search the color name table for *name*, and make it the current name for it's own index.

ARGUMENTS

db	the database node.
index	the index in the color palette.
name	the current name for the given color palette entry.

RETURNS

mgbool	returns **mgTRUE** on success; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgDeleteColorName, **mgNewColorName**, **mgGetCurrentColorName**.

COPYRIGHT

mgPrintRec

NAME

mgPrintRec - prints all the information in a record.

SYNOPSIS

#include lapiinfo.h

void mgPrintRec (mgrec* rec);

DESCRIPTION

mgPrintRec prints all the information associated with a record *rec* to stdout.

ARGUMENTS

rec a record pointer.

RETURNS

None

ACCESS LEVEL

Level 1

SEE ALSO

mgPrintField

COPYRIGHT

mgPrintField

NAME

mgPrintField - prints the information in a value record.

SYNOPSIS

#include lapiinfo.h

void mgPrintField (mgrec* rec);

DESCRIPTION

mgPrintField prints the information associated with a value record *rec* to *stdout*.

ARGUMENTS

rec a value record pointer.

RETURNS

None

ACCESS LEVEL

Level 1

SEE ALSO

mgPrintRec

COPYRIGHT

mgSendError

NAME

mgSendError - reports an error message to *stdout*

SYNOPSIS

#include lapiinfo.h

mgbool mgSendError (char *siteid, int errorcode, ...);

DESCRIPTION

mgSendError looks up the error message corresponding to *errorcode* in a site's message file, resolves the given arguments, prints the resolved message to *stdout*, and sets the resolved message as the last error. The site message file is determined from *siteid*.

ARGUMENTS

siteid	the site's unique identifier as defined in **mgInitDD.**
errorcode	the error index within the message file.

RETURNS

mgbool	returns **mgTRUE** if successful; **mgFALSE** if the message file cannot be found or if the *errorcode* cannot be found in the site's message file.

ACCESS LEVEL

Level 1

SEE ALSO

mgSetError, **mgGetError**, **mgSendWarning**, **mgSendStatus**.

COPYRIGHT

mgGetError

NAME

mgGetError - reports an error message to *stdout*

SYNOPSIS

#include lapiinfo.h

mgbool mgGetError (char **siteid, char **message);

DESCRIPTION

mgGetError returns a pointer to the string containing the *site id* that caused the last error or returns a warning and a pointer to the string containing the last error or warning *message*. These are pointers to the actual *site id* and *message* strings, and should not be changed or deallocated directly by the user.

ARGUMENTS

siteid	a pointer to the site identifier character pointer
message	a pointer to the error message text character pointer.

RETURNS

siteid	returns the site identifier character pointer.
message	returns the error message text character pointer.
mgbool	returns **mgTRUE** if successful; **mgFALSE** if there is no last error.

ACCESS LEVEL

Level 1

SEE ALSO

mgSendError, **mgSetError**, **mgSendWarning**, **mgSendStatus**.

COPYRIGHT

mgSendStatus

NAME

mgSendStatus - reports a status message to *stdout*

SYNOPSIS

#include lapiinfo.h

mgbool mgSendStatus (char *siteid, int code, ...);

DESCRIPTION

mgSendStatus looks up the status message corresponding to *code* in a site's message file, resolves the given arguments, and prints the resolved message to *stdout*. The site message file is determined from *siteid*.

ARGUMENTS

siteid	the site's unique identifier, as defined in **mgInitDD.**
errorcode	the message index within the message file.

RETURNS

mgbool	**mgTRUE** if successful; **mgFALSE** if the message file cannot be found, or if the *code* cannot be found in the site's message file.

ACCESS LEVEL

Level 1

SEE ALSO

mgSendError, **mgGetError**, **mgSendWarning**, **mgSetError**.

COPYRIGHT

mgSendWarning

NAME

mgSendWarning - reports an error message to *stdout*

SYNOPSIS

#include lapiinfo.h

mgbool mgSendWarning (char *siteid, int code, ...);

DESCRIPTION

mgSendWarning looks up the warning message corresponding to *code* in a site's message file, resolves the given arguments, prints the resolved message to *stdout*, and sets the resolved message as the last error. The site message file is determined from *siteid*.

ARGUMENTS

siteid	the site's unique identifier as defined in **mgInitDD.**
errorcode	the index within the message file.

RETURNS

mgbool	**mgTRUE** if successful; **mgFALSE** if the message file cannot be found, or the *code* cannot be found in the site's message file.

ACCESS LEVEL

Level 1

SEE ALSO

mgSendError, **mgGetError**, **mgSetError**.

COPYRIGHT

mgSetError

NAME

mgSetError - sets the last error message

SYNOPSIS

#include lapiinfo.h

mgbool mgSetError (char *siteid, int errorcode, ...);

DESCRIPTION

mgSetError looks up the error message corresponding to *errorcode* in a site's message file, resolves the given arguments, and sets the resolved message as the last error. This routine does not print the error message. The site message file is determined from *siteid*.

ARGUMENTS

siteid	the site's unique identifier as defined in **mgInitDD.**
errorcode	the error index within the message file.

RETURNS

mgbool	returns **mgTRUE** if successful; **mgFALSE** if the message file cannot be found, or if the *errorcode* cannot be found in the site's message file.

EXAMPLE

```
if ( !UserFunc ( uservar ) ) {
    mgSetError (  "USERSITE", 100, "UserFunc" ); /* assumes code 100 requires a string arg. */
    return;
}
```

ACCESS LEVEL

Level 1

SEE ALSO

mgSendError, **mgGetError**, **mgSendWarning**, **mgSendStatus**.

COPYRIGHT

mgCloseDb

NAME

mgCloseDb - closes an opened or newly-created database.

SYNOPSIS

#include lapiio.h

mgbool mgCloseDb (mgrec* db);

DESCRIPTION

mgCloseDb closes the database with the given top node of *db*. This routine is the recommended way to close a database. Besides the system call to close a file, it also performs many housekeeping functions such as updating the time stamp, updating the file revision, and releasing memory.

ARGUMENTS

db a pointer to the top node of the database.

RETURNS

mgbool returns **mgTRUE** on success, otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 1

SEE ALSO

mgOpenDb , **mgWriteDb**

COPYRIGHT

mgExit

NAME

mgExit - exits the MultiGen API software development environment.

SYNOPSIS

#include lapiio.h

mgbool mgExit (void);

DESCRIPTION

mgExit is the very last routine an application should call. Based on the API level, this routine performs the proper exit functions, such as closing all database and resource files, housekeeping, and releasing memory. It is a mandatory call to exit the entire programming environment. **mgExit** returns **mgTRUE** if successful; otherwise **mgFALSE.**

ARGUMENTS

None.

RETURNS

mgbool returns **mgTRUE** if successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 1

COPYRIGHT

mgInit

NAME

mgInit - initializes the MultiGen software development environment.

SYNOPSIS

#include lapiio.h

void mgInit (int* argc, char* argv []);

DESCRIPTION

mgInit is the very first routine an application should call. Based on the API level, this routine performs the proper initialization to the memory, color, material, etc. It is a mandatory call to activate the entire programming environment.

ARGUMENTS

argc	pointer to an *int* containing the number of command line arguments.
argv	the array of command line argument strings

RETURNS

None.

ACCESS LEVEL

Level 1

COPYRIGHT

mgNewDb

NAME

mgNewDb - creates a new database.

SYNOPSIS

#include lapiio.h

mgrec* mgNewDb (char* name);

DESCRIPTION

Given a file name as the string *name*, **mgNewDb** creates a new database associated with this file name. The new database consists of a database node, which is returned via a record pointer.

ARGUMENTS

name the name of the new database file.

RETURNS

mgrec* returns a pointer to top node of the new database; otherwise returns **mgNULL**.

EXAMPLE

```
mgrec *db1, *db2;
db1 = mgOpenDb ( "file1.flt" );
db2 = mgNewDb ( "newfile.flt" );
   /* update 2 databases */
mgWriteDb ( db1 );
mgWriteDb ( db2 );
```

ACCESS LEVEL

Level 2

SEE ALSO

mgOpenDb, **mgWriteDb**, **mgCloseDb**

COPYRIGHT

mgOpenDb

NAME

mgOpenDb - opens an existing OpenFlight database for reading or writing.

SYNOPSIS

#include lapiio.h

mgrec* mgOpenDb (char* filename);

DESCRIPTION

mgOpenDb opens the file named by *filename*. It checks for legal file type and revision, builds the internal bead tree, and returns a pointer to the top node upon successful completion.

ARGUMENTS

filename the name of the file to be opened.

RETURNS

mgrec* returns a pointer to the top node of the database (the database node); **mgNULL** if failed.

EXAMPLE

```
mgrec *db1, *db2;
db1 = mgOpenDb ( "file1.flt" );
db2 = mgNewDb ( "newfile.flt" );
   /* update 2 databases */
mgWriteDb ( db1 );
mgWriteDb ( db2 );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgCloseDb, **mgNewDb**

COPYRIGHT

mgSaveAsDb

NAME

mgSaveAsDb - Writes out a database to disk with a different name than the name with which it was opened.

SYNOPSIS

#include lapiio.h

mgbool mgSaveAsDb (mgrec* db, char *filename);

DESCRIPTION

Given the top node of a database, *db*, and a *filename* **mgSaveAsDb** writes this database to disk using the filename. The given *filename* is stored in the database record as the new file name.

ARGUMENTS

db	the pointer to the top of the database to write.
filename	the new file name.

RETURNS

mgbool	returns **mgTRUE** on success; otherwise returns **mgFALSE**.

EXAMPLE

```
mgrec *db1;
db1 = mgOpenDb ( "file1.flt" );
 mgSaveAsDb ( db1, "newname.flt" );
```

ACCESS LEVEL

Level 2

SEE ALSO

mgOpenDb, **mgNewDb**, **mgCloseDb**, **mgWriteDb**

COPYRIGHT

mgWriteDb

NAME

mgWriteDb - Writes out a database to disk.

SYNOPSIS

#include lapiio.h

mgbool mgWriteDb (mgrec* db);

DESCRIPTION

Given the top node of a database, *db*, **mgWriteDb** writes this database to disk using the file name stored in the database header. The file name must first be opened and initialized with either the **mgOpenDb** or **mgNewDb** functions, which associate the database with a file name.

ARGUMENTS

db the pointer to the top of the database to write.

RETURNS

mgbool returns **mgTRUE** on success; otherwise returns **mgFALSE**.

EXAMPLE

```
mgrec *db1, *db2;
db1 = mgOpenDb ( "file1.flt" );
db2 = mgNewDb ( "newfile.flt" );
   /* update 2 databases */
mgWriteDb ( db1 );
mgWriteDb ( db2 );
```

ACCESS LEVEL

Level 2

SEE ALSO

mgOpenDb, **mgNewDb**, **mgCloseDb**

COPYRIGHT

mgGetFirstLightSource

NAME

mgGetFirstLightSource - gets the first entry record from a database's light source palette.

SYNOPSIS

#include lapilight.h

int mgGetFirstLightSource (mgrec* db, mgrec* ltsrec);

DESCRIPTION

Given a database record *db*, **mgGetFirstLightSource** gets the database's first light source entry record. The record is returned in *ltsrec*, and it is assumed that space has been allocated for the record by a call to **mgNewRec.** The index of the light source in the table is returned if there is one. In the OpenFlight data dictionary, **fltLightSourcePalette** is the typecode to use to create an entry in the light source palette. Do not confuse this with **fltLightSource**, which is a node record.

ARGUMENTS

db	the database node.
ltsrec	the pointer to a pre-allocated light source table entry.

RETURNS

ltsrec	returns the pointer to the light source entry record, if there is one.
int	returns the index of the light source if there is one; otherwise returns **-1**.

ACCESS LEVEL

Level 1

EXAMPLE

```
mgrec *ltsrec;
mgrec *nextrec;
mgrec *db;
int index;
db = mgNewDb ( "newfile.flt" );
ltsrec = mgNewRec ( fltLightSourcePalette );
nextrec = mgNewRec ( fltLightSourcePalette );
index = mgGetFirstLightSource ( db, ltsrec );
index = mgGetNextLightSource ( ltsrec, nextrec );
```

SEE ALSO

mgGetNextLightSource, **mgGetLightSource**

COPYRIGHT

mgGetLightSource

NAME

mgGetLightSource - gets a light source palette entry record from a database's light source palette.

SYNOPSIS

#include lapilight.h

mgbool mgGetLightSource (mgrec* db, int index, mgrec* rec);

DESCRIPTION

mgGetLightSource gets light source entry record *index* from the light source table of *db*. The record is returned in *rec*, and it is assumed that space has been allocated for the record by a call to **mgNewRec.** In the OpenFlight data dictionary, **fltLightSourcePalette** is the typecode to use to create an entry in the light source palette. Do not confuse this with **fltLightSource**, which is a node record.

ARGUMENTS

db	the database node.
index	the table index of the defined light source table entry.
rec	the pointer to a pre-allocated light source table entry.

RETURNS

mgbool	returns **mgTRUE** if light source entry *index* exists; otherwise returns **mgFALSE.**
rec	returns the light source table entry record.

ACCESS LEVEL

Level 1

EXAMPLE

```
int lighttype;
mgrec *lts_rec, *db;
db = mgOpenDb ( "anyfile.flt" );
lts_rec = mgNewRec ( fltLightSourcePalette );
if ( mgGetLightSource ( db, 5, lts_rec ) ) {
  mgGetAttList ( lts_rec, fltLtspType, *lighttype );
}
```

SEE ALSO

mgGetLightSourceCount, **mgGetFirstLightSource**, **mgGetNextLightSource**

COPYRIGHT

mgGetLightSourceCount

NAME

mgGetLightSourceCount - gets the number of entries in a database's light source palette.

SYNOPSIS

#include lapilight.h

int mgGetLightSourceCount (mgrec* db);

DESCRIPTION

mgGetLightSourceCount gets the number of light source entries for a given database *db***.**

ARGUMENTS

db	the database node.

RETURNS

int	returns the number of light source entries.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetFirstLightSource, **mgGetNextLightSource**

COPYRIGHT

mgGetNextLightSource

NAME

mgGetNextLightSource - gets the next entry from a database's light source palette.

SYNOPSIS

#include lapilight.h

int mgGetNextLightSource (mgrec* ltsrec mgrec* nextrec);

DESCRIPTION

Given a light source record, *ltsrec*, **mgGetNextLightSource** fills *nextrec* with the light source record following *ltsrec* in the light source palette. The record is returned in *nextrec*, and it is assumed that space has been allocated for the record by a call to **mgNewRec.** The index of the next light source in the palette is returned, if there is one. In the OpenFlight data dictionary, **fltLightSourcePalette** is the typecode to use to create an entry in the light source palette. Do not confuse this with **fltLightSource**, which is a node record.

ARGUMENTS

ltsrec	the pointer to the light source palette entry.
nextrec	the pointer to the pre-allocated light source palette entry.

RETURNS

nextrec	returns the pointer to the next light source entry record, if there is one.
int	returns the index of the next light source, if there is one; otherwise returns **-1**.

ACCESS LEVEL

Level 1

EXAMPLE

```
mgrec ltsrec;
mgrec nextrec;
mgrec *db;
int index;
db = mgNewDb ( "newfile.flt" );
nextrec = mgNewRec ( fltLightSourcePalette );
index = mgGetFirstLightSource ( db, &ltsrec );
index = mgGetNextLightSource ( ltsrec, &nextrec );
```

SEE ALSO

mgGetFirstLightSource, **mgGetLightSource**

COPYRIGHT

mgIndexOfLightSource

NAME

mgIndexOfLightSource - gets the index of a named light source palette entry.

SYNOPSIS

#include lapilight.h

int mgIndexOfLightSource (mgrec* db, char *name);

DESCRIPTION

mgIndexOfLightSource returns the index of the light source entry record named *name* in the light source palette of *db*. If the named light source is not found, **-1** is returned.

ARGUMENTS

db	the database node.
name	the name of the light source entry to search for.

RETURNS

int	returns the index of the named light source palette entry, or **-1** if no entry is found.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetLightSource, **mgNameOfLightSource**

COPYRIGHT

mgNameOfLightSource

NAME

mgNameOfLightSource - gets the name of a light source palette entry.

SYNOPSIS

#include lapilight.h

char* mgNameOfLightSource (mgrec* db, int index);

DESCRIPTION

mgNameOfLightSource returns a pointer to the name of light source entry record *index* in the light source palette of *db*. If the light source with that index is not found, **mgNULL** is returned. Storage for the name is dynamically allocated by **mgNameOfLightSource**.

Note: the user is responsible for deallocating the dynamically allocated memory using **mgFree**.

ARGUMENTS

db	the database node.
index	the index of the light source entry.

RETURNS

char*	returns a pointer to the light source entry name; otherwise returns **mgNULL**.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetLightSource, **mgIndexOfLightSource**

COPYRIGHT

mgAddLightSource

NAME

mgAddLightSource - allocates and inserts a new light source palette entry record.

SYNOPSIS

#include lapilight.h

int mgAddLightSource (mgrec *db, mgrec **lts_rec, char *name);

DESCRIPTION

Given a database node *db***, mgAddLightSource** allocates a light source record with the *name* given and puts it in the light source palette. The light source record is returned to the user through the light source record pointer *lts_rec*. This record can then be filled with the desired data**.** The index of the entry in the light source palette is returned. The returned light source record is of type **fltLightSourcePalette**. Do not confuse this type with **fltLightSource**, which is a node record.

ARGUMENTS

db	the database node pointer.
lts_rec	the pointer to the new light source entry record pointer.
name	the name of the light source.

RETURNS

int	returns the index of the returned light source in the light source palette, if successful, **-1** if unsuccessful.
lts_rec	returns the light source record pointer.

EXAMPLE

```
mgrec *db, *lts_rec, *spec_rec;
int index;
db = mgOpenDb ( "file1.flt" );
index = mgAddLightSource ( db, &lts_rec, "RedSpotLight" );
mgSetAttList ( lts_rec, fltLtspType, 2,
                        fltLtspSpecular.fltColorR, 1.0,
                        fltLtspSpecular.fltColorG, 0.0,
                        fltLtspSpecular.fltColorB, 0.0,
                mgNULL );
```

ACCESS LEVEL

Level 2

SEE ALSO

mgGetLightSource

COPYRIGHT

mgNewLightSource

NAME

mgNewLightSource - allocates a new record for a light source palette entry.

SYNOPSIS

#include lapilight.h

int mgNewLightSource (mgrec *db, mgrec *lts_rec, char *name);

DESCRIPTION

Given a database node *db***, mgNewLightSource** allocates a light source record with the *name* given and put it in the light source palette. The light source record is returned to the user through the light source record pointer *lts_rec*. This record can then be filled with the desired data**.** The index of the entry in the light source palette is returned. The light source palette entry record is of type **fltLightSourcePalette**. Do not confuse this type with **fltLightSource**, which is a node record.

ARGUMENTS

db	the database node pointer.
lts_rec	the pointer to the new light source record.
name	the name of the light source.

RETURNS

int	returns the index of the returned light source in the light source palette, if successful; **-1** if unsuccessful.
lts_rec	returns the light source record.

EXAMPLE

```
mgrec ltsrec, nextrec;
mgrec *db;
db = mgNewDb ( "newfile.flt" );
mgNewLightSource ( db, ltsrec, "RedLightSource" );
nextrec = mgNewRec ( fltLightSourcePalette );
index = mgGetFirstLightSource ( db, &ltsrec );
index = mgGetNextLightSource ( ltsrec, &nextrec );
```

ACCESS LEVEL

Level 2

SEE ALSO

mgGetLightSource

COPYRIGHT

mgWriteLightSourceFile

NAME

mgWriteLightSourceFile - writes a light source palette as a disk file.

SYNOPSIS

#include lapilight.h

mgbool mgWriteLightSourceFile(mgrec* db, char *fname);

DESCRIPTION

Given a database node, *db*, and a file name *fname***, mgWriteLightSourceFile** writes the database's light source palette to disk with the given file name.

ARGUMENTS

db	the top node of the database.
fname	the name of the light source palette file.

RETURNS

mgbool	returns **mgTRUE** if the file was written successfully; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

COPYRIGHT

mgDelMaterial

NAME

mgDelMaterial - deletes an entry, defined by the material's index, from a database's material palette.

SYNOPSIS

#include lapimaterial.h

mgbool mgDelMaterial (mgrec* db, int index);

DESCRIPTION

mgDelMaterial deletes the entry defined by the material's *index* from the material palette associated with database node *db*. If no material entry is found matching *index*, the material palette remains unchanged.

ARGUMENTS

db	the database node.
index	the index of material to delete.

RETURNS

mgbool	returns **mgTRUE** if *db* contains a valid database; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgDelMaterialByName, **mgIndexOfMaterial**

COPYRIGHT

mgDelMaterialByName

NAME

mgDelMaterialByName - delete a material entry from a database's material palette.

SYNOPSIS

#include lapimaterial.h

mgbool mgDelMaterialByName (mgrec* db, char *name);

DESCRIPTION

mgDelMaterialByName deletes the material entry defined by *name* from the material palette associated with database node *db*. If no material entry is found matching *name*, the material palette remains unchanged.

ARGUMENTS

db	the database node.
name	the name of the material to delete.

RETURNS

mgbool	returns **mgTRUE** if *db* contains a valid database; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgDelMaterial, **mgNameOfMaterial**

COPYRIGHT

mgGetFirstMaterial

NAME

mgGetFirstMaterial - gets the first entry from a database's material palette.

SYNOPSIS

#include lapimaterial.h

int mgGetFirstMaterial (mgrec* db, mgrec* mat_rec);

DESCRIPTION

Given a database record pointer *db*, **mgGetFirstMaterial** gets the first entry record from the database's material palette. The index of the material record is returned in *mat_rec*, and it is assumed that space has been allocated for the record by a call to **mgNewRec**.

ARGUMENTS

db	the database node.
mat_rec	a pointer to a pre-allocated material palette entry record.

RETURNS

int	returns the index of the returned material record, or **-1** if there is none.
mat_rec	returns the first material palette entry record.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetNextMaterial, **mgGetMaterialCount**

COPYRIGHT

mgGetMaterial

NAME

mgGetMaterial - gets an entry record, defined by the entry's *index*, from a database's material palette.

SYNOPSIS

#include lapimaterial.h

mgbool mgGetMaterial (mgrec* db, int index, mgrec* rec);

DESCRIPTION

mgGetMaterial gets material entry record *index* from the material table of database node *db*. The record is returned in *rec*, and it is assumed that space has been allocated for the record by a call to **mgNewRec**.

ARGUMENTS

db	the database node.
index	the index of the defined material palette entry.
rec	a pointer to a pre-allocated material palette entry record.

RETURNS

mgbool	returns **mgTRUE** if material entry *index* exists; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetFirstMaterial, **mgGetNextMaterial**, **mgNewMaterial**

COPYRIGHT

mgGetMaterialCount

NAME

mgGetMaterialCount - gets the number of entries in a database's material palette.

SYNOPSIS

#include lapimaterial.h

int mgGetMaterialCount (mgrec* db);

DESCRIPTION

Given a database node, *db*, **mgGetMaterialCount** gets the number of entries in the database's material palette**.**

ARGUMENTS

db the database node.

RETURNS

int returns the number of entries in the database's material palette.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetFirstMaterial, **mgGetNextMaterial**

COPYRIGHT

mgGetMaterialElem

NAME

mgGetMaterialElem - gets the components of the material used by a record.

SYNOPSIS

#include lapimaterial.h

mgbool mgGetMaterialElem (mgrec* rec, float *ambred, float *ambgreen, float *ambblue, float *difred, float *difgreen, float *difblue, float *spered, float *spegreen, float *speblue, float *emired, float *emigreen, float *emiblue, *shine, float *alpha);

DESCRIPTION

Given a record with a material, *rec***, mgGetMaterialElem** gets the material components of the record's material.

ARGUMENTS

rec	the record pointer.
ambred	the pointer for the ambient red value.
ambgreen	the pointer for the ambient green value.
ambblue	the pointer for the ambient blue value.
difred	the pointer for the diffuse red value.
difgreen	the pointer for the diffuse green value.
difblue	the pointer for the diffuse blue value.
spered	the pointer for the specular red value.
spegreen	the pointer for the specular green value.
speblue	the pointer for the specular blue value.
emired	the pointer for the emissive red value.
emigreen	the pointer for the emissive green value.
emiblue	the pointer for the emissive blue value.
dif	the pointer for the diffuse value.
spe	the pointer for the specular value.
emi	the pointer for the emissivity value.
shine	the pointer for the shininess value.
alpha	the pointer for the alpha (transparency) value.

RETURNS

mgbool	returns **mgTRUE** if the material components are found; returns **mgFALSE** if the record doesn't have material properties.
ambred	returns the ambient red value.
ambgreen	returns the ambient green value.
ambblue	returns the ambient blue value.
difred	returns the diffuse red value.
difgreen	returns the diffuse green value.
difblue	returns the diffuse blue value.
spered	returns the specular red value.
spegreen	returns the specular green value.

speblue	returns the specular blue value.
emired	returns the emissive red value.
emigreen	returns the emissive green value.
emiblue	returns the emissive blue value.
shine	returns the shininess value (0.0-1.0).
alpha	returns the alpha (transparency) value (0.0-1.0).

ACCESS LEVEL

Level 1

SEE ALSO

mgGetFirstMaterial, **mgGetNextMaterial**

COPYRIGHT

mgGetNextMaterial

NAME

mgGetNextMaterial - gets the next material palette entry record from the material palette.

SYNOPSIS

#include lapimaterial.h

int mgGetNextMaterial (mgrec* mat_rec, mgrec* next_rec);

DESCRIPTION

Given a material record pointer *mat_rec*, **mgGetNextMaterial** gets the next material entry record from the material palette. The index of the next material record is returned in *next_rec*, and it is assumed that space has been allocated for the record by a call to **mgNewRec**.

ARGUMENTS

mat_rec	a pointer to a material palette entry.
next_rec	a pointer to a pre-allocated material entry record.

RETURNS

int	returns the index of the returned material record, or **-1** if there is none.
next_rec	returns the next material palette entry.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetFirstMaterial, **mgGetMaterialCount**

COPYRIGHT

mgIndexOfMaterial

NAME

mgIndexOfMaterial - gets the index of a named material palette entry.

SYNOPSIS

#include lapimaterial.h

int mgIndexOfMaterial (mgrec* db, char *name);

DESCRIPTION

mgIndexOfMaterial returns the index of the material palette entry record named *name* in the material table of database node *db*. If the named material is not found, **-1** is returned.

ARGUMENTS

db	the database node.
name	the name of the material entry to search for.

RETURNS

int	returns the index of the named material palette entry, or **-1** if no entry is found.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetMaterial, **mgNewMaterial**

COPYRIGHT

mgNameOfMaterial

NAME

mgNameOfMaterial - gets the name of a material palette entry.

SYNOPSIS

#include lapimaterial.h

char* mgNameOfMaterial (mgrec* db, int index);

DESCRIPTION

mgNameOfMaterial returns the pointer to the name of the material entry record *index* in the material palette of database node *db*. If the material with that index is not found, **mgNULL** is returned. Storage for the name is dynamically allocated by **mgNameOfMaterial**.

Note: the user is responsible for deallocating the dynamically allocated memory with **mgFree**.

ARGUMENTS

db	the database node.
index	the index of the material palette entry.

RETURNS

char*	returns a pointer to the name of the material, if a material name was found; otherwise returns **mgNULL.**

ACCESS LEVEL

Level 1

SEE ALSO

mgGetMaterial, **mgIndexOfMaterial**, **mgNewMaterial**

COPYRIGHT

mgNewMaterial

NAME

mgNewMaterial - Allocate a new record for an entry into a database's material palette.

SYNOPSIS

#include lapimaterial.h

int mgNewMaterial (mgrec* db, mgrec* ent_rec, char *name);

DESCRIPTION

mgNewMaterial allocates a record containing a new material table entry, adds it to the database's material palette, and passes back the record in *ent_rec*.

ARGUMENTS

db	the database node.
ent_rec	a pointer to the record that contains the new material.
name	the name for the new material (may be **mgNULL).**

RETURNS

int	returns the index of the new material palette entry.
ent_rec	returns the new material record.

ACCESS LEVEL

Level 2

SEE ALSO

mgDelMaterial

COPYRIGHT

mgWriteMaterialFile

NAME

mgWriteMaterialFile - writes a database's material palette as a disk file.

SYNOPSIS

#include lapimaterial.h

mgbool mgWriteMaterialFile (mgrec* db, char *fname);

DESCRIPTION

Given a database node pointer, *db*, **mgWriteMaterialFile** writes the database's material palette to a file with the name *fname***.**

ARGUMENTS

db	the record pointer to the top node of the database.
fname	the material palette file name.

RETURNS

mgbool	returns **mgTRUE** if the material palette file was written successfully; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

COPYRIGHT

mgMatrixFormRotate

NAME

mgMatrixFormRotate - given direction cosines, forms a rotation matrix for rotation around a vector.

SYNOPSIS

#include lapimath.h

void mgMatrixFormRotate (mgMatrix *m, VTYPE theta, VTYPE a, VTYPE b, VTYPE c);

DESCRIPTION

Given a vector defined by *a*, *b*, and *c*, and an angular value *theta*, **mgMatrixFormRotate** forms a rotation matrix defined by the vector and angle, and returns it in the matrix that has the pointer *m*. It is assumed that the matrix pointed to by *m* has been pre-allocated.

ARGUMENTS

m	the pointer for the returned matrix.
theta	the angular value.
a, b, c	the values that define the vector.

RETURNS

m	returns the pointer to the returned rotation matrix**.**

ACCESS LEVEL

Level 1

COPYRIGHT

mgTransformCoord

NAME

mgTransformCoord - transforms a coordinate by applying a transformation matrix.

SYNOPSIS

#include lapimath.h

icoord mgTransformCoord (mgMatrix m, icoord *coord);

DESCRIPTION

Given a transformation matrix *m*, and a coordinate *coord*, **mgTransformCoord** transforms the coordinate by applying the matrix. The resulting coordinate is returned.

ARGUMENTS

m	the transformation matrix.
coord	the coordinate.

RETURNS

icoord	returns the transformed coordinate.

ACCESS LEVEL

Level 1

SEE ALSO

mgAddCoord

COPYRIGHT

mgAddCoord

NAME

mgAddCoord - adds two coordinates.

SYNOPSIS

#include lapimath.h

icoord mgAddCoord (icoord *coord1, icoord *coord2);

DESCRIPTION

Given two coordinates, *icoord1* and *icoord2*, **mgAddCoord** adds the coordinates and returns the resulting coordinate.

ARGUMENTS

coord1	the first coordinate.
coord2	the second coordinate.

RETURNS

icoord	returns the resulting coordinate.

EXAMPLE

ACCESS LEVEL

Level 1

SEE ALSO

mgTransformCoord

COPYRIGHT

mgVectorFromLine

NAME

mgVectorFromLine - forms a vector from a line.

SYNOPSIS

#include lapimath.h

dvector mgVectorFromLine (dline *line);

DESCRIPTION

Given the line, *line*, **mgVectorFromLine** returns the vector formed by subtracting the line's first coordinate from its second coordinate.

ARGUMENTS

line the line.

RETURNS

dvector returns the resulting vector.

ACCESS LEVEL

Level 1

COPYRIGHT

mgUnitizeVector

NAME

mgUnitizeVector - changes a vector to a unit vector.

SYNOPSIS

#include lapimath.h

void mgUnitizeVector (dvector *v);

DESCRIPTION

Given a vector *v*, **mgUnitizeVector** changes *v* to a unit vector by dividing each component of the vector by the vector's magnitude.

ARGUMENTS

v the vector.

RETURNS

v returns the unit vector.

ACCESS LEVEL

Level 1

SEE ALSO

mgMakeVector, **mgVectorFromLine**

COPYRIGHT

mgDistance

NAME

mgDistance - gets the distance between two coordinates.

SYNOPSIS

#include lapimath.h

VTYPE mgDistance (icoord *coord1, icoord *coord2);

DESCRIPTION

Given two coordinates *coord1* and *coord2*, **mgDistance** returns the distance between them.

ARGUMENTS

coord1	the first coordinate.
coord2	the second coordinate.

RETURNS

VTYPE	returns the distance between the two given coordinates.

ACCESS LEVEL

Level 1

SEE ALSO

mgMakeVector

COPYRIGHT

mgMakeVector

NAME

mgMakeVector - makes a vector from two coordinates.

SYNOPSIS

#include lapimath.h

dvector mgMakeVector (icoord *coord1, icoord *coord2);

DESCRIPTION

Given two coordinates *coord1* and *coord2*, **mgMakeVector** returns the vector based at *coord1* and in the direction of *coord2*.

ARGUMENTS

coord1	the first coordinate.
coord2	the second coordinate.

RETURNS

dvector	returns the vector made from the two given coordinates.

ACCESS LEVEL

Level 1

SEE ALSO

mgVectorFromLine, **mgUnitizeVector**

COPYRIGHT

mgCrossProdVector

NAME

mgCrossProdVector - gets the cross-product of two vectors.

SYNOPSIS

#include lapimath.h

dvector mgCrossProdVector (dvector *vec1, dvector *vec2);

DESCRIPTION

Given two vectors, *vec1* and *vec2*, **mgCrossProdVector** returns the cross-product of the two vectors.

ARGUMENTS

vec1	the first vextor.
vec2	the second vector.

RETURNS

dvector	returns the cross-product vector of the two given vectors.

ACCESS LEVEL

Level 1

SEE ALSO

mgMakeVector, **mgUnitizeVector**

COPYRIGHT

mgMalloc

NAME

mgMalloc - allocates a dynamic memory buffer.

SYNOPSIS

#include lapimath.h

void *mgMalloc (unsigned int byte_size);

DESCRIPTION

Given the number of bytes to allocate, *byte_size*, **mgMalloc** returns a pointer to a memory buffer of that size. All of the bytes of the buffer are preset to zero. The data type of the returned buffer is a void pointer. The calling function should use a **cast** statement to convert the buffer to the desired data type. When the buffer is no longer needed, use the **mgFree** function to deallocate it.

ARGUMENTS

byte_size is the number of bytes to allocate.

RETURNS

void* returns the pointer to the allocated and zeroed buffer.

ACCESS LEVEL

Level 1

SEE ALSO

mgFree

COPYRIGHT

mgFree

NAME

mgFree - deallocates a dynamic memory buffer.

SYNOPSIS

#include lapimath.h

void mgFree (void *buf);

DESCRIPTION

Given a pointer to a dynamic memory buffer, *ptr*, **mgFree** deallocates it. Use **mgMalloc** to allocate dynamic memory buffers. It is a serious error to attempt to free a memory block that is already deallocated; in this case, **mgFree** displays the message “attempt to free a free block @<*address*>.”

ARGUMENTS

buf the pointer to the dynamic memory buffer.

ACCESS LEVEL

Level 1

SEE ALSO

mgMalloc

COPYRIGHT

mgAppend

NAME

mgAppend - links a node record into the database hierarchy.

SYNOPSIS

#include lapistruc.h

mgbool mgAppend (mgrec* parentrec, mgrec* childrec);

DESCRIPTION

Given a parent node *parentrec*, mgAppend links a node record, *childrec*, and its children into the database hierarchy. The resulting hierarchy has *childrec* as the **last** child of *parentrec*.

ARGUMENTS

parentrec	the parent node record.
childrec	the node record to be inserted into the database hierarchy.

RETURNS

mgbool	returns **mgTRUE** if the attachment was successful; otherwise returns **mgFALSE**.

EXAMPLE

```
mgrec *db, *par_rec, *poly1, *poly2, *poly3;
db = mgOpenDb ( "file1.flt" );
par_rec = GetParentNodeFunc ( db );
poly1 = mgNewRec ( fltPolygon ); poly2 = mgNewRec ( fltPolygon ); poly3 = mgNewRec ( fltPolygon );
mgAttach ( par_rec, poly1 ); /* poly1 is first child of par_rec */
mgAppend ( par_rec, poly2 ); /* poly2 is the last child of par_rec */
mgInsert ( poly2, poly3 ); /* poly3 is the next-to-last child of par_rec */
```

ACCESS LEVEL

Level 2

SEE ALSO

mgAttach, **mgInsert**

COPYRIGHT

mgAttach

NAME

mgAttach - links a node record into the database hierarchy.

SYNOPSIS

#include lapistruc.h

mgbool mgAttach (mgrec* parentrec, mgrec* childrec);

DESCRIPTION

Given a parent node record *parentrec*, **mgAttach** links a node record, *childrec*, and its children into the database hierarchy. The resulting hierarchy has *childrec* as the **first** child of *parentrec*.

ARGUMENTS

parentrec	the parent node record.
childrec	the node record to be inserted into the database hierarchy.

RETURNS

mgbool	returns **mgTRUE** if the attachment is successful; otherwise returns **mgFALSE.**

EXAMPLE

```
mgrec *db, *par_rec, *poly1, *poly2, *poly3;
db = mgOpenDb ( "file1.flt" );
par_rec = GetParentNodeFunc ( db );
poly1 = mgNewRec ( fltPolygon ); poly2 = mgNewRec ( fltPolygon ); poly3 = mgNewRec (fltPolygon);
mgAttach ( par_rec, poly1 ); /* poly1 is first child of par_rec */
mgAppend ( par_rec, poly2 ); /* poly2 is the last child of par_rec */
mgInsert ( poly2, poly3 ); /* poly3 is the next-to-last child of par_rec */
```

ACCESS LEVEL

Level 2

SEE ALSO

mgAppend, **mgInsert**

COPYRIGHT

mgCountChild

NAME

mgCountChild - gets the number of children of a node record.

SYNOPSIS

#include lapistruc.h

int mgCountChild (mgrec* noderec);

DESCRIPTION

mgCountChild returns the number of children directly below a node record *noderec*. Descendents of the children are not counted in the total.

ARGUMENTS

noderec a node record.

RETURNS

int returns the number of child node records immediately below the given node record.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetChildNth

COPYRIGHT

mgDelete

NAME

mgDelete - deletes a given record.

SYNOPSIS

#include lapistruc.h

mgbool mgDelete (mgrec* rec);

DESCRIPTION

Given a record *rec*, **mgDelete** deletes this record. The record can be one data element or a set of elements, including other such records. If the record is a node in the database, and has children, its children are recursively deleted as well.

ARGUMENTS

rec points to the record to delete.

RETURNS

mgbool returns **mgTRUE** on success; otherwise **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgDetach

COPYRIGHT

mgDeReference

NAME

mgDeReference - makes an existing instance node into a regular node.

SYNOPSIS

#include lapistruc.h

mgbool mgDeReference (mgrec* rec);

DESCRIPTION

mgDeReference removes the instance relationship between *rec* and its reference node.

ARGUMENTS

rec	an instance node record.

RETURNS

mgbool	returns **mgTRUE** if the instance unlinking is successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgReference

COPYRIGHT

mgDetach

NAME

mgDetach - unlinks a node record from the database hierarchy.

SYNOPSIS

#include lapistruc.h

mgbool mgDetach (mgrec* rec);

DESCRIPTION

mgDetach unlinks a node record, *rec*, and its children from the database hierarchy. The node is left as an orphan, and is not deleted.

ARGUMENTS

rec a node record.

RETURNS

mgbool returns **mgTRUE** if the detachment was successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgDelete

COPYRIGHT

mgDuplicate

NAME

mgDuplicate - duplicates a node in the database.

SYNOPSIS

#include lapistruc.h

mgrec* mgDuplicate (mgrec* rec);

DESCRIPTION

Given a pointer to the node record *rec* in a database, **mgDuplicate** duplicates this node record, returning a pointer to the new (duplicate) record.

ARGUMENTS

rec a pointer to the node to duplicate.

RETURNS

mgrec* returns a pointer to the new node; returns **mgNULL** if failed.

ACCESS LEVEL

Level 2

SEE ALSO

mgCopyRec

COPYRIGHT

mgNewRec

NAME

mgNewRec - allocates a new node record.

SYNOPSIS

#include lapistruc.h

mgrec *mgNewRec (mgcode rcode);

DESCRIPTION

Given a typecode *rcode*, **mgNewRec** allocates and initializes a record of the type defined by *rcode*. A pointer to the newly created node record is returned. If the create is unsuccessful, **mgNULL** is returned. The newly created record is an orphan, meaning it does not have a place in a database hierarchy.

ARGUMENTS

rcode the typecode for the new node record.

RETURNS

mgrec* returns the pointer to the newly created node record; **mgNULL** if unsuccessful.

EXAMPLE

```
mgrec *db, *par_rec, *poly1, *poly2, *poly3;
db = mgOpenDb ( "file1.flt" );
par_rec = GetParentNodeFunc ( db );
poly1 = mgNewRec ( fltPolygon ); poly2 = mgNewRec ( fltPolygon ); poly3 = mgNewRec ( fltPolygon );
mgAttach ( par_rec, poly1 ); /* poly1 is first child of par_rec */
mgAppend ( par_rec, poly2 ); /* poly2 is the last child of par_rec */
mgInsert ( poly2, poly3 ); /* poly3 is the next-to-last child of par_rec */
```

ACCESS LEVEL

Level 2

SEE ALSO

mgDuplicate, **mgDelete**, **mgAttach**, **mgAppend**, **mgInsert**, **mgReference**

COPYRIGHT

mgGetChildNth

NAME

mgGetChildNth - gets a node from a parent node's children list.

SYNOPSIS

#include lapistruc.h

mgrec* mgGetChildNth (mgrec* noderec, int nth);

DESCRIPTION

mgGetChildNth will return the *nth* child node in the children list of a given parent node *noderec*. The first child is indicated when *nth* = 0.

ARGUMENTS

noderec	a parent node record.
nth	which child to return from the list of children.

RETURNS

mgrec*	A record pointer to the child node found; returns **mgNULL** if failed.

EXAMPLE

```
mgrec *db, *node_rec, *par_rec, *child_rec;
int i;
db = mgOpenDb ( "file1.flt" );
node_rec = GetNodeRecFunc ( db );
par_rec = mgGetParent ( node_rec );
for ( i=0 ; child_rec = mgGetChildNth ( par_rec, i ) ; i++ ) {
    if ( child_rec == node_rec ) {
        printf ( "node_rec is the %dth child of par_rec", i+1 );
        return;
    }
}
printf ( "node_rec is not a child of par_rec" );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgCountChild, mgGetChild, mgGetNext, mgGetParent

COPYRIGHT

mgGetNext
mgGetNext, mgGetPrevious, mgGetParent, mgGetChild, mgGetNestedParent, mgGetNestedChild, mgGetReference

NAME

mgGetNext, **mgGetPrevious**, **mgGetParent**, **mgGetChild**, **mgGetNestedParent**, **mgGetNestedChild**, **mgGetReference** - find a node record's adjacent node in an OpenFlight database.

SYNOPSIS

#include lapistruc.h

mgrec* mgGetNext (mgrec* rec);
mgrec* mgGetPrevious (mgrec* rec);
mgrec* mgGetParent (mgrec* rec);
mgrec* mgGetChild (mgrec* rec);
mgrec* mgGetNestedParent (mgrec* rec);
mgrec* mgGetNestedChild (mgrec* rec);
mgrec* mgGetReference (mgrec* rec);

DESCRIPTION

These are the functions to find the adjacent node of a given node in the OpenFlight hierarchical database. **mgGetNext** returns *rec*'s next node at the same hierarchy level. **mgGetPrevious** returns *rec*'s previous node. **mgGetParent** gets its parent node, and **mgGetChild** gets the first child node of *rec*'s sibling list. **mgGetNestedParent** and **mgGetNestedChild** return the nested parent and child node of a polygon node, respectively. **mgGetReference** gets the reference node, which is being shared by other nodes.

ARGUMENTS

rec a node record.

RETURNS

mgrec* returns the adjacent node record; returns **mgNULL** if failed.

EXAMPLE

```
mgrec *db, *node_rec, *next_rec, *prev_rec, *par_rec, *child_rec, *ref_rec;
db = mgOpenDb ( "file1.flt" );
node_rec = GetNodeRecFunc ( db );
next_rec = mgGetNext ( node_rec );
prev_rec = mgGetPrevious ( node_rec );
par_rec = mgGetParent ( node_rec );
child_rec = mgGetChild ( node_rec );
if ( !child_rec ) /* can't have both child and reference */
   ref_rec = mgGetReference ( node_rec );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgWalk.

COPYRIGHT

mgGetFirstInstance
mgGetNextInstance

NAME

mgGetFirstInstance, **mgGetNextInstance** - get the instances of a reference node.

SYNOPSIS

#include lapistruc.h

mgrec *mgGetFirstInstance (mgrec *ref);

mgrec *mgGetNextInstance (mgrec *inst);

DESCRIPTION

Instancing is a technique in which one node is a child of several parent nodes. The child is said to be a "reference" and each parent is an "instance" of the child. A parent node can have either a regular child or a reference child, but not both. Given a reference node *ref*, **mgGetFirstInstance** returns the first instance (the first node in the list of nodes that reference the reference node). If *ref* is not a reference node, **mgNULL** is returned. Given an instance node *inst*, **mgGetNextInstance** returns the next node in the instance list.

ARGUMENTS

ref	a reference node record.
inst	an instance node record.

RETURNS

mgrec*	returns the first or next instance of the reference node record.

EXAMPLE

```
mgrec *ref;
mgrec *inst;
inst = mgGetFirstInstance ( ref );
while ( inst ) {
  action( inst );
  inst = mgGetNextInstance ( inst );
}
```

ACCESS LEVEL

Level 1

SEE ALSO

mgIsFirstInstance, **mgReference**, **mgDeReference**

COPYRIGHT

mgIsFirstInstance

NAME

mgIsFirstInstance - finds out if a record is the first instance of its referenced record.

SYNOPSIS

#include lapistruc.h

mgbool mgIsFirstInstance (mgrec* rec);

DESCRIPTION

Given an existing node *rec*, **mIsFirstInstance** returns **mgTRUE** if *rec* is the first instance of its referenced node; **mgFALSE** otherwise. This routine is useful when used with **mgGetFirstInstance** and **mgGetNextInstance** to visit all instances of a record, or when used by itself to visit only the first instance.

ARGUMENTS

rec a node record with a link to a reference record.

RETURNS

mgbool returns **mgTRUE** if *rec* is the first instance of its referenced node; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgReference, **mgDeReference**, **mgGetFirstInstance**, **mgGetNextInstance**

COPYRIGHT

mgAddMatrix

NAME

mgAddMatrix - adds a matrix to a node record.

SYNOPSIS

#include lapiutil.h

mgbool mgAddMatrix (mgrec* rec, mgMatrix* matrix);

DESCRIPTION

If the node record *rec* already has a matrix, then **mgAddMatrix** is multiplied by the existing matrix, *matrix;* otherwise *matrix* becomes the new matrix for *rec.*

ARGUMENTS

rec	a node record.
matrix	the new matrix for *rec.*

RETURNS

mgbool	returns **mgTRUE** on success; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgUpdateMatrix

COPYRIGHT

mgGetMatrix

NAME

mgGetMatrix - gets a copy of the matrix of a node record.

SYNOPSIS

#include lapiutil.h

mgbool mgGetMatrix (mgrec* noderec, mgMatrix *matrix);

DESCRIPTION

Given a node record *noderec* and storage for an **mgMatrix** *matrix*, **mgGetMatrix** copies the matrix of *noderec* into *matrix.*

ARGUMENTS

noderec	a node record.
matrix	the address of an **mgMatrix**

RETURNS

matrix	a copy of the matrix for the node record *noderec.*
mgbool	**mgTRUE** if successful, **mgFALSE** if *noderec* has no matrix, or otherwise unsuccessful.

ACCESS LEVEL

Level 1

SEE ALSO

mgUpdateMatrix

COPYRIGHT

mgGetRepCount

NAME

mgGetRepCount - gets the replication count of a node record.

SYNOPSIS

#include lapiutil.h

int mgGetRepCount (mgrec* rec);

DESCRIPTION

Given the node record *rec*, **mgGetRepCount** returns the number of replications the record has. Replication is the operation used to repeat the drawing of one object several times by applying a transformation each time.

ARGUMENTS

rec	the pointer to the node record.

RETURNS

int	returns the number of replications of the node record.

ACCESS LEVEL

Level 1

COPYRIGHT

mgIsFlagOn

NAME

mgIsFlagOn - determines if the given node record's ON flag is set.

SYNOPSIS

#include lapiutil.h

mgbool mgIsFlagOn (mgrec *rec);

DESCRIPTION

Given the node record, *rec*, **mgIsFlagOn** determines if this record's ON flag is set. The ON flag is used by the Level of Detail routines to distinguish between different levels of detail. The current level of detail always has the ON flag set.

ARGUMENTS

rec a pointer to a node record.

RETURNS

mgbool returns **mgTRUE** if the given node record's ON flag is set; otherwise **mgFALSE**.

ACCESS LEVEL

Level 1

SEE ALSO

mgMoreDetail, **mgLessDetail**, **mgMostDetail**, **mgLeastDetail**, **mgWalk**

COPYRIGHT

mgIsInstance

NAME

mgIsInstance - determines if the given structure record is an instance (has a reference).

SYNOPSIS

#include lapiutil.h

mgbool mgIsInstance (mgrec *rec);

DESCRIPTION

Given the record *rec*, **mgIsInstance** determines if this record is an instance; that is, whether it references a shared record (or reference record).

ARGUMENTS

rec the pointer to the record in question.

RETURNS

mgbool returns **mgTRUE** if the record is an instance; otherwise returns **mgFALSE**.

SEE ALSO

mgGetReference, **mgReference**, **mgDeReference**, **mgIsReference**, **mgIsFirstInstance**, **mgGetFirstInstance**, **mgGetNextInstance**

ACCESS LEVEL

Level 2

COPYRIGHT

mgIsReference

NAME

mgIsReference - determines if the given node record is a reference (shared).

SYNOPSIS

#include lapiutil.h

mgbool mgIsReference (mgrec *rec);

DESCRIPTION

Given the record, *rec*, **mgIsReference** determines if this record is a reference; that is, whether it is shared or referenced by other node records.

ARGUMENTS

rec the pointer to the record in question.

RETURNS

mgbool returns **mgTRUE** if the record is a reference; otherwise returns **mgFALSE**.

SEE ALSO

mgGetReference, **mgReference**, **mgDeReference**, **mgIsInstance**, **mgIsFirstInstance**, **mgGetFirstInstance**, **mgGetNextInstance**

ACCESS LEVEL

Level 2

COPYRIGHT

mgCopyRec

NAME

mgCopyRec - copies the contents of a record.

SYNOPSIS

#include lapistruc.h

void mgCopyRec (mgrec *from, mgrec *to);

DESCRIPTION

Given a record pointer *from*, **mgCopyRec** uses **memcpy** to copy the contents of the record to the location specified by another given record pointer, *to*. It is assumed that sufficient storage has been allocated to *to*.

ARGUMENTS

from	the source record pointer.
to	the destination record pointer.

ACCESS LEVEL

Level 1

COPYRIGHT

mgGetRecByName

NAME

mgGetRecByName - finds a node from the given node ID.

SYNOPSIS

#include lapistruc.h

mgrec* mgGetRecByName (mgrec* db, char* name);

DESCRIPTION

Given a string *name*, **mgGetRecByName** searches the given database node, *db*, and locates the node in the database that has *name* as its ID. Each node has a unique case-sensitive name, hence there is a one-to-one correspondence between the input name and the returned node.

ARGUMENTS

root	the database node.
name	the node ID that is the object of the search.

RETURNS

mgrec*	returns the node when found; otherwise returns **mgNULL**.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetName

COPYRIGHT

mgHasXform

NAME

mgHasXform - checks whether a node record has a transformation associated with it.

SYNOPSIS

#include lapistruc.h

mgbool mgHasXform (mgrec* noderec);

DESCRIPTION

mgHasXform checks a node record *noderec* for transformations, and returns **mgTRUE** if a transformation is found.

ARGUMENTS

noderec a node record.

RETURNS

mgbool returns **mgTRUE** if there is at least one transformation associated with *noderec*; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 1

SEE ALSO

mgGetXform, **mgGetXformType**

COPYRIGHT

mgGetXform

NAME

mgGetXForm - gets a transformation record.

SYNOPSIS

#include lapistruc.h

mgrec *mgGetXform (mgrec *rec);

DESCRIPTION

Given a node record *rec*, **mgGetXform** returns its transformation record.

ARGUMENTS

rec a node record.

RETURNS

mgrec* returns the transformation record.

ACCESS LEVEL

Level 1

SEE ALSO

mgHasXform, **mgGetXformType**

COPYRIGHT

mgGetXformType

NAME

mgGetXformType - gets the instances of a reference record.

SYNOPSIS

#include lapistruc.h

XFLL_opcodesE mgGetXformType (mgrec *ref);

DESCRIPTION

Instancing is a technique in which one record can be a child of several parent records. The child is said to be a “reference” and each parent is an “instance” of the child. A parent record can have either a regular child or a reference child, but not both. Given a reference node *ref*, **mgGetFirstInstance** returns the first instance (the first record in the list of records that reference the reference node). If *ref* is not a reference node, **mgNULL** is returned. Given an instance record *inst*, **mgGetNextInstance** returns the next record in the instance list.

ARGUMENTS

ref	a reference node record.
inst	an instance node record.

RETURNS

mgrec*	returns the first/next instance of the reference node record.

EXAMPLE

```
mgrec *ref;
mgrec *inst;
inst = mgGetFirstInstance ( ref );
while ( inst ) {
  action( inst );
  inst = mgGetNextInstance ( inst );
}
```

ACCESS LEVEL

Level 1

SEE ALSO

mgIsFirstInstance, **mgReference**, **mgDeReference**

COPYRIGHT

mgInsert

NAME

mgInsert - links a node record into the database hierarchy.

SYNOPSIS

#include lapistruc.h

mgbool mgInsert (mgrec* prevrec, mgrec* childrec);

DESCRIPTION

Given a previous sibling node *prevrec*, **mgInsert** links a node record *childrec* and its children into the database hierarchy. The resulting hierarchy has *childrec* as the next sibling of *prevrec*.

ARGUMENTS

prevrec	the node record of the previous sibling.
childrec	the node record to be inserted into the database hierarchy.

RETURNS

mgbool	returns **mgTRUE** if the attachment was successful; otherwise returns **mgFALSE**.

EXAMPLE

```
mgrec *db, *par_rec, *poly1, *poly2, *poly3;
db = mgOpenDb ( "file1.flt" );
par_rec = GetParentNodeFunc ( db );
poly1 = mgNewRec ( fltPolygon ); poly2 = mgNewRec ( fltPolygon ); poly3 = mgNewRec ( fltPolygon );
mgAttach ( par_rec, poly1 ); /* poly1 is first child of par_rec */
mgAppend ( par_rec, poly2 ); /* poly2 is the last child of par_rec */
mgInsert ( poly2, poly3 ); /* poly3 is the next-to-last child of par_rec */
```

ACCESS LEVEL

Level 2

SEE ALSO

mgAppend, **mgAttach**

COPYRIGHT

mgReference

NAME

mgReference - changes an existing node record into an instance node record.

SYNOPSIS

#include lapistruc.h

mgbool mgReference (mgrec* rec, mgrec* ref_rec);

DESCRIPTION

Given an existing node record *rec*, **mgReference** causes *rec* to become an instance of the reference node record *ref_rec*.

ARGUMENTS

rec	a node record.
ref_rec	a reference node record.

RETURNS

mgbool	Returns **mgTRUE** if the instancing was successful; otherwise returns **mgFALSE.**

ACCESS LEVEL

Level 2

SEE ALSO

mgDeReference

COPYRIGHT

mgWalk

NAME

mgWalk - walks the database hierarchy tree and performs an action at each node.

SYNOPSIS

#include lapistruc.h

mgbool mgWalk (mgrec* noderec, mgwalk_func preaction, mgwalk_func postaction, void *data, int flags)

DESCRIPTION

mgWalk iterates from any node *noderec* down the hierarchy and visits the nodes within the tree using a depth-first search. The types of nodes visited are based on traversal modifiers specified in *flags*. At each node visited, two user-provided functions (*preaction* and *postaction*) are invoked to perform necessary tasks. Preaction is invoked before the node's children are traversed, and postaction is invoked after the node's children have been traversed. As an option, the traversal can be terminated by either the *preaction* or *postaction* function returning **mgFALSE**.

ARGUMENTS

noderec the starting node record of the traversal.

preaction, postaction
the function pointers to user functions defined as:
mgbool (*mgwalk_func) (mgrec* db_rec, mgrec* parent_rec, mgrec *this_rec, void *data)
where *db_rec* is the database record, *parent_rec* is the parent node of the current node record, and *this_rec* is the current node being visited. **mgWalk** terminates instantly if a **mgFALSE** is returned from the preaction routine. The preaction routine is invoked BEFORE any of the node's children are visited. The postaction routine is invoked AFTER all the node's children have been visited. Null *preaction* and *postaction* values are valid.

data the pointer to user data that is passed through to the *preaction* and *postaction* routines.

flags traversal modifiers that allow the user to visit only certain types of nodes.

MGWALKNEXT - also visit siblings of *noderec* and their children. The default is to not visit siblings.

MGWALKVERTEX - also visit vertex nodes. The default is to not visit vertex nodes.

MGWALKMASTER - also visit each referenced node only once. The default is to not visit referenced nodes.

MGWALKMASTERALL - also visit each referenced node once for each time it is referenced. The default is to not visit referenced nodes.

MGWALKON - visit only nodes that are part of the current level of detail. The default is to visit the nodes of all Levels of Detail.

MGWALKNORDONLY - do not visit nodes that are read-only (external nodes, for example). The default is to visit read-only nodes.

RETURNS

mgbool returns **mgTRUE** upon successful completion, returns **mgFALSE** if unsuccessful or if terminated by one of the action functions.

ACCESS LEVEL

Level 1

COPYRIGHT

mgMoreDetail

NAME

mgMoreDetail - changes a database's Level of Detail (LOD) to the next higher level.

SYNOPSIS

#include lapistruc.h

mgbool mgMoreDetail (mgrec* db)

DESCRIPTION

mgMoreDetail sorts the switch-in distances from the LOD records in database record *db*, retrieves the database's current switch-in distance, changes the current switch-in value to the next lower distance (the next higher LOD), and unsets the *on* flag for each of the LOD records' children, except for ones with the new current switch-in distance. Subsequent calls change the current switch-in value further, and set the LOD records' *on* flags. The return value is **mgFALSE** if there is no higher Level of Detail. If the database does not have any LOD records, the first call to **mgMoreDetail** returns **mgTRUE**, while subsequent calls return **mgFALSE**.

ARGUMENTS

db	a database's top node.

RETURNS

mgbool	returns **mgTRUE** upon successful completion; returns **mgFALSE** if there is no higher Level of Detail.

EXAMPLE

```
mgLeastDetail ( db );
while ( mgMoreDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
mgMostDetail ( db ); /* should already be at most LOD */
while (mgLessDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgLessDetail, **mgMostDetail**, **mgLeastDetail**, **mgWalk**

COPYRIGHT

mgLessDetail

NAME

mgLessDetail - changes a database's Level of Detail (LOD) to the next lower level.

SYNOPSIS

#include lapistruc.h

mgbool mgLessDetail (mgrec* db)

DESCRIPTION

mgLessDetail sorts the switch-in distances from all LOD records in database *db*, retrieves the database's current switch-in distance, changes the current switch-in value to the next higher distance (the next lower LOD), and unsets the *on* flag for each of the LOD records' children, except for ones with the new current switch-in distance. Subsequent calls change the current switch-in value further and set the LOD records' *on* flags. The return value is **mgFALSE** if there is no lower Level of Detail. If the database has does not have any LOD records, the first call to **mgLessDetail** returns **mgTRUE**, while subsequent calls return **mgFALSE**.

ARGUMENTS

db a database's top record.

RETURNS

mgbool returns **mgTRUE** upon successful completion; returns **mgFALSE** if there is no lower level of detail.

EXAMPLE

```
mgLeastDetail ( db );
while ( mgMoreDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
mgMostDetail ( db ); /* should already be at most LOD */
while (mgLessDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgMoreDetail, mgLeastDetail, mgMostDetail, mgWalk

COPYRIGHT

mgMostDetail

NAME

mgMostDetail - changes a database's Level of Detail (LOD) to the highest level.

SYNOPSIS

#include lapistruc.h

mgbool mgMostDetail (mgrec* db)

DESCRIPTION

mgMostDetail sorts the switch-in distances from all LOD records in database record *db*, changes the current switch-in value to the lowest distance (the highest LOD), and unsets the *on* flag for each of the LOD records' children, except for ones with the lowest switch-in distance. If there are no LOD records or if there is only one LOD record in the database, **mgMostDetail** does nothing and returns **mgTRUE**, since the database is already at the most detail.

ARGUMENTS

db a database's top record.

RETURNS

mgbool returns **mgTRUE** upon successful completion. Otherwise, **mgFALSE**

EXAMPLE

```
mgLeastDetail ( db );
while ( mgMoreDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
mgMostDetail ( db ); /* should already be at most LOD */
while (mgLessDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgMoreDetail, **mgLessDetail**, **mgLeastDetail**, **mgWalk**

COPYRIGHT

mgLeastDetail

NAME

mgLeastDetail - changes a database's Level of Detail (LOD) to the lowest level.

SYNOPSIS

#include lapistruc.h

mgbool mgLeastDetail (mgrec* db)

DESCRIPTION

mgLeastDetail sorts the switch-in distances from all LOD records in database record *db*, changes the current switch-in value to the highest distance (the lowest LOD), and unsets the *on* flag for each of the LOD records' children, except for ones with the highest switch-in distance. If the database does not have any LOD records, or if there is only one LOD record, **mgLeastDetail** does nothing and returns **mgTRUE**, since the database is already at the least detail.

ARGUMENTS

db a database's top record.

RETURNS

mgbool returns **mgTRUE** upon successful completion, **mgFALSE** otherwise.

EXAMPLE

```
mgLeastDetail ( db );
while ( mgMoreDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
mgMostDetail ( db ); /* should already be at most LOD */
while (mgLessDetail ( db ) )
  mgWalk ( db, PrintEachNodeName, mgNULL, mgNULL, ILON );
```

ACCESS LEVEL

Level 1

SEE ALSO

mgLessDetail, **mgMoreDetail**, **mgMostDetail**, **mgWalk**

COPYRIGHT

mgRec2Filename

NAME

mgRec2Filename - gets the database file name from any record of a database.

SYNOPSIS

#include lapistruc.h

char *mgRec2Filename (mgrec *rec);

DESCRIPTION

Given any record in a database, *rec*, **mgRec2Filename** returns the database file name. If a record is not associated with any database, a null string is returned. The file name returned has a path if it was opened or created in a remote directory. Storage for the file name is dynamically allocated by **mgRec2Filename**.

Note: the user is responsible for deallocating the dynamically allocated memory with **mgFree**.

ARGUMENTS

rec	a pointer to any record.

RETURNS

char*	returns the file name; **mgNULL** if no record is found.

EXAMPLE

ACCESS LEVEL

Level 1

SEE ALSO

mgGetCurrentDb, **mgRec2Db**

COPYRIGHT

mgRec2Db

NAME

mgRec2Db - gets the database file name from any record of a database.

SYNOPSIS

#include lapistruc.h

mgrec *mgRec2Db (mgrec *rec);

DESCRIPTION

Given any record in a database, *rec*, **mgRec2db** returns a pointer to the top node of the database. If a record is not associated with any database, the top node of the current database is returned.

ARGUMENTS

rec a pointer to any record.

RETURNS

mgrec* returns the pointer to the top node of the database.

EXAMPLE

```
mgrec *poly_rec, *db1, *db2, *cur_db;
db1 = mgOpenDb ( "file1.flt" );
db2 = mgOpenDb ( "file2.flt" );
poly_rec = GetPolyRecFunc ( db1 );
cur_db = mgGetCurrentDb();
if ( cur_db != mgRec2Db ( poly_rec ) )
   printf ("poly_rec not in current database");
```

ACCESS LEVEL

Level 1

SEE ALSO

mgGetCurrentDb, **mgRec2Filename**

COPYRIGHT

mgSetCurrentDb

NAME

mgSetCurrentDb - sets a database as the current database.

SYNOPSIS

#include lapistruc.h

void mgSetCurrentDb (mgrec *rec);

DESCRIPTION

Given any record in a database, *rec*, **mgSetCurrentDb** sets this database as the current database. If the record is a new current database, the color palette associated with this database is loaded.

ARGUMENTS

rec a pointer to any record in a database.

RETURNS

None.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetCurrentDb, **mgRec2Db**

COPYRIGHT

mgGetCurrentDb

NAME

mgGetCurrentDb - gets the current database.

SYNOPSIS

#include lapistruc.h

mgrec *mgGetCurrentDb (void);

DESCRIPTION

Returns a pointer to the top node of the current database. If there is no current database, returns **mgNULL**.

ARGUMENTS

RETURNS

mgrec*	is a pointer to the top node of the current database. Returns **mgNULL** if there is no current database.

EXAMPLE

```
mgrec *poly_rec, *db1, *db2, *cur_db;
db1 = mgOpenDb ( "file1.flt" );
db2 = mgOpenDb ( "file2.flt" );
poly_rec = GetPolyRecFunc ( db1 );
cur_db = mgGetCurrentDb();
if ( cur_db != mgRec2Db ( poly_rec ) )
   printf ("poly_rec not in current database");
```

ACCESS LEVEL

Level 1

SEE ALSO

mgSetCurrentDb, **mgRec2Db**

COPYRIGHT

mgIsCode

NAME

mgIsCode - determines if the given node record is a given type.

SYNOPSIS

#include lapiutil.h

mgbool mgIsCode (mgrec *rec, mgcode type);

mgbool mgIsBsp (mgrec *rec);

mgbool mgIsDof (mgrec *rec);

mgbool mgIsGroup (mgrec *rec);

mgbool mgIsHeader (mgrec *rec);

mgbool mgIsLightSource (mgrec *rec);

mgbool mgIsLod (mgrec *rec);

mgbool mgIsObject (mgrec *rec);

mgbool mgIsPolygon (mgrec *rec);

mgbool mgIsSound (mgrec *rec);

mgbool mgIsSwitch (mgrec *rec);

mgbool mgIsPath (mgrec *rec);

mgbool mgIsVertex (mgrec *rec);

DESCRIPTION

Given the node record, *rec*, **mgIsCode** determines if this record is of type, *type*.

The others are convenience functions which determine if a node record is of a pre-determined type:

mgIsBsp - fltBsp.

mgIsDof - fltDof.

mgIsGroup - fltGroup.

mgIsHeader - fltHeader.

mgIsLightSource - fltLightSource.

mgIsLod - fltLod.

mgIsObject - fltObject.

mgIsPolygon - fltPolygon.

mgIsSound - fltSound.

mgIsSwitch - fltSwitch.

mgIsPath - fltPath.

mgIsVertex - fltVertex.

ARGUMENTS

rec	a pointer to a node record.
type	an OpenFlight node record type.

RETURNS

mgbool	returns **mgTRUE** if the node record is of the specified type, otherwise **mgFALSE**.

ACCESS LEVEL

Level 1

COPYRIGHT

mgGetVtxNormal

NAME

mgGetVtxNormal - retrieves the normal vector from a vertex record.

SYNOPSIS

#include lapiutil.h

mgbool mgGetVtxNormal (mgrec* rec, float* i, float* j, float* k);

DESCRIPTION

mgGetVtxNormal retrieves the *i*, *j*, and *k* values of the vertex normal vector from a vertex record *rec*.

ARGUMENTS

rec	a vertex record.
i	a pointer to a float for storing the *i* component of the normal vector.
j	a pointer to a float for storing the *j* component of the normal vector.
k	a pointer to a float for storing the *k* component of the normal vector.

RETURNS

mgbool	returns **mgTRUE** if the vertex has a normal vector; otherwise returns **mgFALSE.**
i	the *i* component of the normal vector.
j	the *j* component of the normal vector.
k	the *k* component of the normal vector.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetPolyNormal

COPYRIGHT

mgGetPolyNormal

NAME

mgGetPolyNormal - retrieves the normal vector from a polygon record.

SYNOPSIS

#include lapiutil.h

mgbool mgGetPolyNormal (mgrec* rec, double *i, double *j, double *k);

DESCRIPTION

mgGetPolyNormal retrieves the *i*, *j*, and *k* values of the polygon normal vector from a polygon vertex record *rec*.

ARGUMENTS

rec	a vertex record.
i	a pointer to a double for storing the *i* component of the normal vector.
j	a pointer to a double for storing the *j* component of the normal vector.
k	a pointer to a double for storing the *k* component of the normal vector.

RETURNS

mgbool	returns **mgTRUE** if the polygon has a normal vector; otherwise returns **mgFALSE.**
i	returns the *i* component of the normal vector.
j	returns the *j* component of the normal vector.
k	returns the *k* component of the normal vector.

ACCESS LEVEL

Level 1

SEE ALSO

mgGetVtxNormal

COPYRIGHT

mgIsPolyConcave

NAME

mgIsPolyConcave - determines if the given polygon is concave.

SYNOPSIS

#include lapiutil.h

mgbool mgIsPolyConcave (mgrec *poly);

DESCRIPTION

Given the polygon record *poly*, **mgIsPolyConcave** determines if this polygon is concave. If there is any line on the plane of the polygon that crosses more than two of the edges of the polygon, then the polygon is concave.

ARGUMENTS

poly a pointer to the polygon record.

RETURNS

mgbool returns **mgTRUE** if the polygon is coplanar; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 1

COPYRIGHT

mgIsCoplanar

NAME

mgIsCoplanar - determines if the given polygon is coplanar.

SYNOPSIS

#include lapiutil.h

mgbool mgIsCoplanar (mgrec *poly);

DESCRIPTION

Given the record *poly*, containing a polygon node in a database, **mgIsCoplanar** determines if this polygon is coplanar. If all of the vertices of this polygon lie in the same plane, then the polygon is coplanar.

ARGUMENTS

poly a pointer to the record containing a polygon node.

RETURNS

mgbool returns **mgTRUE** if the polygon is coplanar; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

COPYRIGHT

mgParseFname

NAME

mgParseFname - gets the components of a file name (*path*, *filename*, and *extension*).

SYNOPSIS

#include lapiutil.h

int mgParseFname (char *filename, char **pathname, char **fname, char **ext);

DESCRIPTION

Given a file name complete with its path and extension, *filename*, **mgParseFname** returns the path in *pathname*, the file name without its extension in *fname*, and the extension in *ext*. **mgParseFname** returns the length in characters of *fname*. The parsing is done such that *pathname* is all characters from the beginning of the *filename* string to the last "/" (or empty if there are no "/"s), *fname* is all characters after the last "/" and the first "." (or from the beginning if there are no "/"s and to the end if there are no "."s), *ext* is all characters after the first "." (or empty if there are no "."s).

ARGUMENTS

filename	the file name to parse.
pathname	the path string pointer.
fname	the filename (without extension) string pointer
ext	the extension string pointer

RETURNS

pathname	the returned path.
fname	the returned file name without extension.
ext	the returned extension.
int	the length in characters of the returned file name (*fname*).

ACCESS LEVEL

Level 2

COPYRIGHT

mgReverse

NAME

mgReverse - reverses the order of the vertices in a polygon.

SYNOPSIS

#include lapiutil.h

mgbool mgReverse (mgrec *poly);

DESCRIPTION

Given the record, *poly*, containing a polygon node in a database, **mgReverse** reverses the order of the vertices in the polygon. This causes the polygon to face the other direction.

ARGUMENTS

poly a pointer to the record containing a polygon node.

RETURNS

mgbool returns **mgTRUE** on success; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

COPYRIGHT

mgTriangulate

NAME

mgTriangulate - triangulates a given polygon.

SYNOPSIS

#include lapiutil.h

mglist* mgTriangulate (mgrec* poly);

DESCRIPTION

Given the record, *poly*, containing a polygon node in a database, **mgTriangulate** splits this polygon into triangles. The original polygon is left untouched, and a list of new polygon records is returned. This list contains polygons with only three sides.

ARGUMENTS

poly a pointer to the record containing a polygon node.

RETURNS

mglist* returns a list of records containing the triangles made from the polygon; **mgNULL** if failed.

ACCESS LEVEL

Level 2

COPYRIGHT

mgUpdateMatrix

NAME

mgUpdateMatrix - updates the matrix in a node record.

SYNOPSIS

#include lapiutil.h

mgbool mgUpdateMatrix (mgrec* noderec);

DESCRIPTION

mgUpdateMatrix traverses the transformation-linked list of the node record *noderec* and updates the matrix accordingly.

ARGUMENTS

noderec a node record.

RETURNS

mgbool returns **mgTRUE** on success; otherwise returns **mgFALSE**.

ACCESS LEVEL

Level 2

SEE ALSO

mgGetMatrix

COPYRIGHT

mgCoordDif

NAME

mgCoordDif - finds the difference between two coordinates.

SYNOPSIS

#include lapiutil.h

icoord mgCoordDif (icoord *coord1, icoord *coord2);

DESCRIPTION

Given two coordinates, *coord1* and *coord2*, **mgCoordDif** returns a coordinate that has, as its X value, the difference of the two given coordinates' X values. The same applies for its Y and Z values.

ARGUMENTS

coord1	the first coordinate.
coord2	the second coordinate.

RETURNS

icoord	returns the difference coordinate.

ACCESS LEVEL

Level 1

SEE ALSO

mgMakeLine, **mgi2fcoord**.

COPYRIGHT

mgVectorMove

NAME

mgVectorMove - moves a coordinate along a direction vector.

SYNOPSIS

#include lapiutil.h

icoord mgVectorMove (icoord *coord, vector *vec, MLTYPE n);

DESCRIPTION

Given a coordinate *coord*, a direction vector *vec*, and a number of units *n*, **mgDVectorToVector** moves the coordinate *n* units along the vector. If the coordinate is not given, one is created at (0, 0, 0), moved *n* units along the vector, and returned.

ARGUMENTS

coord	the coordinate.
vec	the direction vector.
n	the number of units.

RETURNS

icoord	returns the coordinate (0, 0, 0) moved *n* units and returned if no coordinate is passed in.

ACCESS LEVEL

Level 1

SEE ALSO

mgDVectorToVector.

COPYRIGHT

mgi2fcoord

NAME

mgi2fcoord - gets a single-precision floating point coordinate from an integer coordinate.

SYNOPSIS

#include lapiutil.h

fcoord mgi2fcoord (icoord *coord);

DESCRIPTION

Given an integer coordinate, *coord*, **mgi2fcoord** returns a vector that is *coord* cast as a single-precision floating point coordinate.

ARGUMENTS

coord the integer coordinate.

RETURNS

fcoord returns the single-precision floating point coordinate.

ACCESS LEVEL

Level 1

COPYRIGHT

mgDVectorToVector

NAME

mgDVectorToVector -returns a vector from a given double vector.

SYNOPSIS

#include lapiutil.h

vector mgDVectorToVector (dvector *vec);

DESCRIPTION

Given a double-precision floating point vector *vec*, **mgDVectorToVector** returns a vector, *vec*, which is cast as a single-precision floating point vector.

ARGUMENTS

vec the double-precision floating point vector.

RETURNS

vector returns the single-precision floating point vector.

ACCESS LEVEL

Level 1

COPYRIGHT

mgMakeLine

NAME

mgMakeLine - finds the difference between two coordinates.

SYNOPSIS

#include lapiutil.h

dline mgMakeLine (icoord *coord1, icoord *coord2);

DESCRIPTION

Given two coordinates, *coord1* and *coord2*, **mgMakeLine** returns a line between the two coordinates.

ARGUMENTS

coord1	the first coordinate.
coord2	the second coordinate.

RETURNS

dline	returns the line between the two coordinates.

ACCESS LEVEL

Level 1

SEE ALSO

mgCoordDif, **mgi2fcoord**.

COPYRIGHT

MultiGen OpenFlight API Glossary of Terms

alpha The transparency value of a **texel**. Each texel is assigned an alpha value in the range of 0 (completely transparent) to 255 (completely opaque).

API The MultiGen OpenFlight Application Programming Interface, which is a set of C header files and libraries that provide a programming interface to MultiGen and the OpenFlight database format.

API level One of four levels of the MultiGen OpenFlight API. Level 1:*Read*, gives you access to OpenFlight files. Level 2: *Write*, creates MultiGen node hierarchies and saves them in OpenFlight format. Level 3: *Extensions*, extends the kind of data represented in the OpenFlight format. These extensions then are treated as native data by MultiGen and GameGen. Level 4:*Tools* allows you to define plug-ins to MultiGen. This release of the API includes Levels 1 and 2, Read and Write.

attribute One of the defining properties of a record. Database attributes include flags, and numeric and text data such as color, relative priority, and LOD switching distances.

attribute code The unique integer code, defined in the data dictionary for an attribute, that specifies which attribute record you want to access.

attribute record A record containing descriptive data (flags, names, numeric values, and text) that is saved with a node in the database hierarchy. Node types, texture attributes, light color, and even the OpenFlight format revision number can all be represented as attribute records, which are accessed through attribute codes.

A record that contains a single property, such as .

attribute value The numeric value or setting of an attribute.

bead Another name for a database node.

child node A node attached below another node in the database tree. The child node inherits transformations and other properties from its parent node.

code Rules that define how data is represented, and which can be used to convert data from one format to another.

color palette The indexed set of available colors.

concave polygon A polygon with two or more faces that form an inward indentation. If a line can be drawn between two vertices in a concave polygon, and the line is located outside the edges of the polygon, the polygon is concave.

current color The color to be applied to any object that is in the process of being created, or is about to be created.

current database The .flt file that is open and is in the top window, ready for subsequent operations.

current light source The light source to be applied to any object that is in the process of being created, or is about to be created.

current material The **material** to be applied to any object that is in the process of being created, or is about to be created.

current texture The **texture** to be applied to any object that is in the process of being created, or is about to be created.

database The geometry and hierarchy that defines all models in a .flt file.

database header Def

database node The top or root node of the database hierarchy. It contains the database header information, and through it, the entire database can be accessed.

data dictionary A list of all the files, fields, and variables used in a database management system.

data type The category to which a given set of data belongs. Most API parameters are described by standard C data types: char, short, int, and double.

default color palette The color palette that is not associated with a particular database. It is used when the color palette is accessed, but there is no current database.

entry	One element of a palette, an entry is accessed in its palette by either its index or its name.
extension record	def
external record	def
group	A node that has two or more objects as its children.
header	The record that contains the information for the top node in the database.
image	A two-dimensional matrix of texels.
image attributes	The type, height, and width of an image.
index	An item, usually an entry in a table, that is used to obtain the address of another item of data. A pointer is a type of index.
instance node	A node that references a portion of the geometry in the database. The referenced geometry is called a **reference node**.
instancing	The ability to describe a group or object once, then display it one or more times with various transformations. OpenFlight supports instancing of objects, groups, and group-like nodes, using transformations such as rotate, translate, scale, and put.
internal bead tree	The entire hierarchy of the database, minus external reference beads.
level of detail	One of a set of models that represent the same item in the database at varying levels of complexity. When the eyepoint is far away from the item, a simple (low-resolution) level of detail is displayed. As the eyepoint approaches the item, lower resolution models switch out and increasingly complex (higher resolution) models switch in.
light	Brightness added to the colors of an image.
light source	A point at which light in an image originates. Light shines along a definable direction vector from the light source.
light source index	A pointer to an entry in the light source palette.
light source palette	The indexed set of available light sources.
material	A surface property that simulate the light-reflecting characteristics of substances such as wood, plastic, or metal.
material index	A pointer to an entry in the material palette.
material palette	The indexed set of available materials.
nested node	A node representing a polygon (face) attached to a coplanar face in the hierarchy. There are parent nested nodes and child nested nodes; these are equivalent to MultiGen's superface and subface nodes, respectively. Nested nodes provide a useful means of specifying drawing order on z-buffered systems, because in z-buffered drawing mode, every subface draws on top of its parent.
next node	When traversing the database, the node that is read immediately after the current node.
node	A generic term for any record that forms the hierarchy of the OpenFlight database. Records with typecodes **fltDof, fltGroup, fltHeader, fltObject, fltPolygon, fltSwitch,** and **fltVertex** are nodes.
node attributes	
node type	
object	A node that has two or more coplanar subfaces as its children.
orphan	A node that has no attach point in the hierarchy. Orphans are not saved with the database.
palette	An organized, indexed set of available properties, such as colors, materials, texture images, and light sources, that can be applied to certain elements of the database. A database usually contains one of each type of palette.
palette index	An address or other indicator that gives the location of an item stored in a palette.
parent node	The attach point of any given node in the database.
pixel	The smallest unit of an image on a display screen.

polygon	A multi-sided, closed face consisting of edges that connect vertices.
previous node	When traversing the database, the node that is read immediately before the current node.
record	An organized collection of properties, attributes, and/or links that fully describes an element of an OpenFlight database and potentially its position in the database.Records can contain attributes that are other records, as well as attributes that are simple values. Records that contain only one attribute value are called *value records*. Records that have links to other records are called *nodes*.
record level	def
reference	A link to a shared node (reference node).
reference node	A node that is referenced by more than one other node. This type of node is also called a *shared node*. Each time a node is shared, it is said to be “instanced.”
replicate	Create a copy.
RGB	An image type whose colors are defined by red, green, and blue values.
root node	The **database node**.
schema	The logical organization of a database.
shared node	See **reference node**.
sibling	One of at least two child nodes that are attached at the same level of the database.
site ID	A unique code within a schema that indicates the organization who defined a set of extensions, and the configuration they are going to have. One site ID per data dictionary.
texel	The basic unit of a texture. A texel can include a color value, a transparency (alpha) value, or both.
texture	A bitmapped image used to add the appearance of complexity to a flat surface.
texture index	A pointer to an entry in the texture palette.
texture palette	The indexed set of available textures.
top node	The **database node**.
top record	The **database node**.
transformation	A function that positions one or more nodes in the database without modifying their vertex coordinates. The transformation is applied to all the descendants of the node containing the matrix.
transformation matrix	An array of values that, when applied, position one or more nodes in the database without modifying their vertex coordinates. Transformation matrices are useful with instances and external references, where you want to position a single object in several places, and in functions that reposition database objects and groups, such as translate and scale. You can apply a transformation matrix to any node type but **fltPolygon** and **fltVertex**.
traverse	The order in which beads in the database hierarchy are processed for drawing. In OpenFlight, the hierarchy is traversed from top to bottom and then left to right. This means each bead is processed relative to its neighbors, as follows: The bead’s left sibling is processed. The bead is processed. The bead’s children are processed (from left to right). The bead’s right sibling is processed.
value record	A record that includes only one attribute value.
vertex	One of the coordinate points that defines a polygon.