Blender Class Index

[Index] [Hierarchy]

Object	Blender objects can be seen in the OOPS window as gray rectangles
Mesh	Blender meshes can be seen in the OOPS window as brown rectangles
Material	Blender materials can be seen in the OOPS window as cyan rectangles
Camera	Blender cameras can not be seen in the OOPS window at the moment
DisplaySettings	The display settings are reflected in Blender's display buttons (F10)
Lamp	Blender lamps can be seen in the OOPS window as yellow rectangles
Scene	Blender scenes can be seen in the OOPS window as green rectangles
Blender	This is in fact no class it is a module in Python
GUI	This is in fact no class it is a module in Python
FileSelector	A file selector can be seen in Blender when you load (F1) or save (F2) a file

---

Blender Class Hierarchy

- Blender
  - Camera
  - DisplaySettings
  - Lamp
  - Material
  - Mesh
  - Object
  - Scene
- GUI
  - FileSelector

Blender Class Reference

[Blender Index] [Blender Hierarchy]

This is in fact no class it is a module in Python More...

`#include <blender.h>`

Public Members

- **Blender** ()
- **~Blender** ()
- PyObject* **addMesh** (const char* type)
- PyObject* **connect** (PyObject* obj1, PyObject* obj2)
- PyObject* **getCamera** (const char* name)
- PyObject* **getCurrentScene** ()
- PyObject* **getDisplaySettings** ()
- PyObject* **getLamp** (const char* name)
- PyObject* **getMaterial** (const char* name)
- PyObject* **getMesh** (const char* name)
- PyObject* **getObject** (const char* name)
- PyObject* **isCamera** (const char* name)
- PyObject* **isLamp** (const char* name)
- PyObject* **isMesh** (const char* name)
- PyObject* **setCurrentFrame** (int frame)

Detailed Description

This is in fact no class it is a module in Python. But I use it here as a class to document the global functions availabe in the Blender module.

PyObject* addMesh(const char* type)

This reflects what happens if you press SPACE in Blender and select ADD->Mesh. The type string is one of the strings shown in Blender's submenu ("Plane", "Cube", etc.).

PyObject* connect(PyObject* obj1, PyObject* obj2)

This should connect things like it is shown in the OOPS window. At the moment you can only connect objects with meshes and scenes with objects. This is likely to change in the future.

PyObject* getCamera(const char* name)

Even if you can't see the connected data to an camera object in the OOPS window yet you can get access to the camera settings by calling this function with the name of the camera data.

PyObject* getCurrentScene()

Gives access to the current scene. For export scripts this should be one of the first things you should do. The names of the objects in the scene are stored and you can step through this names and get access to the associated data by calling getObject(name).

PyObject* getDisplaySettings()

Because you can't create instances of the class DisplaySettings you have to call this function to create one for you. Then you can retrieve the data from the returned instance.

PyObject* getLamp(const char* name)

Gives access to the lamp data by using the unique name.

PyObject* getMaterial(const char* name)

Gives access to the material data by using the unique name.

PyObject* getMesh(const char* name)

Gives access to the mesh data by using the unique name.

PyObject* getObject(const char* name)

Gives access to the object data by using the unique name.

PyObject* isCamera(const char* name)

If you want to check if the connected data to this object (with this name) is really a camera use this function.

PyObject* isLamp(const char* name)

If you want to check if the connected data to this object (with this name) is really a lamp use this function.

PyObject* isMesh(const char* name)

If you want to check if the connected data to this object (with this name) is really a mesh use this function.

PyObject* setCurrentFrame(int frame)

If you want to export an animation you have to step through all the frames. This function allows you to do so and updates all objects which have some IPOs (InterPOlation curves) connected.

- *Author*: Jan Walter **K***doc*
- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          blender.h  -  description
                             -------------------
    begin                : Thu Dec 7 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef BLENDER_H
#define BLENDER_H

struct PyObject;

/**This is in fact no class it is a module in Python. But I use it here as a class to
document the global functions availabe in the Blender module.
  *@author Jan Walter
  */

class Blender {
public:
        Blender();
        ~Blender();
  /** This reflects what happens if you press SPACE in Blender and select ADD->Mesh.
The type string is one of the strings shown in Blender's submenu ("Plane", "Cube",
etc.). */
  PyObject* addMesh(const char* type);
  /** This should connect things like it is shown in the OOPS window. At the moment
you can only connect objects with meshes and scenes with objects. This is likely to
change in the future. */
  PyObject* connect(PyObject* obj1, PyObject* obj2);
  /** Even if you can't see the connected data to an camera object in the OOPS window
yet you can get access to the camera settings by calling this function with the name
of the camera data. */
  PyObject* getCamera(const char* name);
  /** Gives access to the current scene. For export scripts this should be one of the
first things you should do. The names of the objects in the scene are stored and you
can step through this names and get access to the associated data by calling
getObject(name). */
  PyObject* getCurrentScene();
  /** Because you can't create instances of the class DisplaySettings you have to
call this function to create one for you. Then you can retrieve the data from the
returned instance. */
  PyObject* getDisplaySettings();
  /** Gives access to the lamp data by using the unique name. */
  PyObject* getLamp(const char* name);
  /** Gives access to the material data by using the unique name. */
```

```
  PyObject* getMaterial(const char* name);
  /** Gives access to the mesh data by using the unique name. */
  PyObject* getMesh(const char* name);
  /** Gives access to the object data by using the unique name. */
  PyObject* getObject(const char* name);
  /** If you want to check if the connected data to this object (with this name) is
really a camera use this function. */
  PyObject* isCamera(const char* name);
  /** If you want to check if the connected data to this object (with this name) is
really a lamp use this function. */
  PyObject* isLamp(const char* name);
  /** If you want to check if the connected data to this object (with this name) is
really a mesh use this function. */
  PyObject* isMesh(const char* name);
  /** If you want to export an animation you have to step through all the frames.
This function allows you to do so and updates all objects which have some IPOs
(InterPOlation curves) connected. */
  PyObject* setCurrentFrame(int frame);
};

#endif
```

Camera Class Reference

[Blender Index] [Blender Hierarchy]

Blender cameras can not be seen in the OOPS window at the moment More...

`#include <camera.h>`

Inherits: Blender

Public Members

- **Camera** ()
- **~Camera** ()
- char* **name**
- PyObject* **Lens**
- PyObject* **ClSta**
- PyObject* **ClEnd**

Detailed Description

Blender cameras can not be seen in the OOPS window at the moment. Only the corresponding object is visible in gray.

---

char* name

All rectangles you can see in the OOPS window are instances of classes with an unique name for all instances of this class. Nevertheless the name of two instances of different classes can be the same.

PyObject* Lens

The lens value of Blender is a bit odd. If you want to calculate the FOV (field of view angle) you should know that fov = 360.0 * math.atan(factor * 16.0 / camera.Lens) / math.pi) where factor is dependend on the x- and y-resolution of your picture.

PyObject* ClSta

Clipping start value

PyObject* ClEnd

Clipping end value

- *Author*: Jan Walter **K***doc*
- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          camera.h  -  description
                             -------------------
    begin                : Thu Dec 7 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef CAMERA_H
#define CAMERA_H

#include <blender.h>

struct PyObject;

/**Blender cameras can not be seen in the OOPS window at the moment. Only the
corresponding object is visible in gray.
  *@author Jan Walter
  */

class Camera : public Blender {
public:
        Camera();
        ~Camera();
private: // Private attributes
  /** All rectangles you can see in the OOPS window are instances of classes with an
unique name for all instances of this class. Nevertheless the name of two instances
of different classes can be the same. */
  char* name;
  /** The lens value of Blender is a bit odd. If you want to calculate the FOV (field
of view angle) you should know that fov = 360.0 * math.atan(factor * 16.0 /
camera.Lens) / math.pi) where factor is dependend on the x- and y-resolution of your
picture. */
  PyObject* Lens;
  /** Clipping start value */
  PyObject* ClSta;
  /** Clipping end value */
  PyObject* ClEnd;
};

#endif
```

DisplaySettings Class Reference

[Blender Index] [Blender Hierarchy]

The display settings are reflected in Blender's display buttons (F10) More...

`#include <displaysettings.h>`

Inherits: Blender

Public Members

- **DisplaySettings** ()
- **~DisplaySettings** ()
- PyObject* **startFrame**
- PyObject* **endFrame**
- PyObject* **currentFrame**
- PyObject* **xResolution**
- PyObject* **yResolution**
- PyObject* **pixelAspectRatio**

Detailed Description

The display settings are reflected in Blender's display buttons (F10).

PyObject* startFrame

Start frame for animations

PyObject* endFrame

End frame for animations

PyObject* currentFrame

Current frame for animations

PyObject* xResolution

The image width in pixels

PyObject* yResolution

The image height in scanlines

PyObject* pixelAspectRatio

Is the same as Blender's AspY / AspX (see display buttons [F10])

```
/***************************************************************************
                          displaysettings.h  -  description
                             -------------------
    begin                : Thu Dec 7 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef DISPLAYSETTINGS_H
#define DISPLAYSETTINGS_H

#include <blender.h>

struct PyObject;

/**The display settings are reflected in Blender's display buttons (F10).
  *@author Jan Walter
  */

class DisplaySettings : public Blender {
public:
        DisplaySettings();
        ~DisplaySettings();
private: // Private attributes
  /** Start frame for animations */
  PyObject* startFrame;
  /** End frame for animations */
  PyObject* endFrame;
  /** Current frame for animations */
  PyObject* currentFrame;
  /** The image width in pixels */
  PyObject* xResolution;
  /** The image height in scanlines */
  PyObject* yResolution;
  /** Is the same as Blender's AspY / AspX (see display buttons [F10]) */
  PyObject* pixelAspectRatio;
};

#endif
```

Lamp Class Reference

[Blender Index] [Blender Hierarchy]

Blender lamps can be seen in the OOPS window as yellow rectangles More...

`#include <lamp.h>`

Inherits: Blender

Public Members

- **Lamp** ()
- **~Lamp** ()
- char* **name**
- PyObject* **type**
- PyObject* **mode**
- PyObject* **Energ**
- PyObject* **R**
- PyObject* **G**
- PyObject* **B**
- PyObject* **Dist**
- PyObject* **SpoSi**
- PyObject* **SpoBl**
- PyObject* **Quad1**
- PyObject* **Quad2**
- PyObject* **HaInt**
- PyObject* **ClipSta**
- PyObject* **ClipEnd**

Detailed Description

Blender lamps can be seen in the OOPS window as yellow rectangles. They are connected to an object. Compare the attributes of a lamp to the settings of the lamp buttons (F4).

---

char* name

All rectangles you can see in the OOPS window are instances of classes with an unique name for all instances of this class. Nevertheless the name of two instances of different classes can be the same.

PyObject* type

The type of a lamp can be: "Lamp", "Spot", "Sun", or "Hemi"

PyObject* mode

The mode indicates with a string of length 8 if a setting in the lamp buttons (F4) is on ("1") or off ("0"). The order is exactly the same as in the lamp buttons (from top downwards): "Quad", "Sphere", "Shadows", "Halo", "Layer", "Negative", "OnlyShadow", and "Square".

PyObject* Energ

Light energy

PyObject* R

Red part of the lamp color

PyObject* G

Green part of the lamp color

PyObject* B

Blue part of the lamp color

PyObject* Dist

Influences the light attenuation

PyObject* SpoSi

Spotlight setting: Spot size (angle)

PyObject* SpoBl

Spotlight setting: Spot blend (falloff from full light intensity to darkness)

PyObject* Quad1

Influences the light attenuation

PyObject* Quad2

Influences the light attenuation

PyObject* HaInt

The intensity of the spot halo

PyObject* ClipSta

Clipping start value

PyObject* ClipEnd

Clipping end value

```
/***************************************************************************
                          lamp.h  -  description
                             -------------------
    begin                : Thu Dec 7 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef LAMP_H
#define LAMP_H

#include <blender.h>

struct PyObject;

/**Blender lamps can be seen in the OOPS window as yellow rectangles. They are
connected to an object. Compare the attributes of a lamp to the settings of the lamp
buttons (F4).
  *@author Jan Walter
  */

class Lamp : public Blender {
public:
        Lamp();
        ~Lamp();
private: // Private attributes
  /** All rectangles you can see in the OOPS window are instances of classes with an
unique name for all instances of this class. Nevertheless the name of two instances
of different classes can be the same. */
  char* name;
  /** The type of a lamp can be: "Lamp", "Spot", "Sun", or "Hemi" */
  PyObject* type;
  /** The mode indicates with a string of length 8 if a setting in the lamp buttons
(F4) is on ("1") or off ("0"). The order is exactly the same as in the lamp buttons
(from top downwards): "Quad", "Sphere", "Shadows", "Halo", "Layer", "Negative",
"OnlyShadow", and "Square". */
  PyObject* mode;
  /** Light energy */
  PyObject* Energ;
  /** Red part of the lamp color */
  PyObject* R;
  /** Green part of the lamp color */
  PyObject* G;
  /** Blue part of the lamp color */
  PyObject* B;
  /** Influences the light attenuation */
```

```
  PyObject* Dist;
  /** Spotlight setting: Spot size (angle) */
  PyObject* SpoSi;
  /** Spotlight setting: Spot blend (falloff from full light intensity to darkness)
*/
  PyObject* SpoBl;
  /** Influences the light attenuation */
  PyObject* Quad1;
  /** Influences the light attenuation */
  PyObject* Quad2;
  /** The intensity of the spot halo */
  PyObject* HaInt;
  /** Clipping start value */
  PyObject* ClipSta;
  /** Clipping end value */
  PyObject* ClipEnd;
};

#endif
```

Material Class Reference

[Blender Index] [Blender Hierarchy]

Blender materials can be seen in the OOPS window as cyan rectangles More...

`#include <material.h>`

Inherits: Blender

Public Members

- **Material** ()
- **~Material** ()
- char* **name**
- PyObject* **R**
- PyObject* **G**
- PyObject* **B**

Detailed Description

Blender materials can be seen in the OOPS window as cyan rectangles. I decided that a list of materials (only their names) are hold by the objects. This is a bit inconsistent with the OOPS window where materials can be connected with objects or meshes for example.

---

char* name

All rectangles you can see in the OOPS window are instances of classes with an unique name for all instances of this class. Nevertheless the name of two instances of different classes can be the same.

PyObject* R

Red part of color

PyObject* G

Green part of color

PyObject* B

Blue part of color

- *Author*: Jan Walter **K***doc*
- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          material.h  -  description
                             -------------------
    begin                : Thu Dec 7 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef MATERIAL_H
#define MATERIAL_H

#include <blender.h>

struct PyObject;

/**Blender materials can be seen in the OOPS window as cyan rectangles. I decided
that a list of materials (only their names) are hold by the objects. This is a bit
inconsistent with the OOPS window where materials can be connected with objects or
meshes for example.
  *@author Jan Walter
  */

class Material : public Blender {
public:
        Material();
        ~Material();
private: // Private attributes
  /** All rectangles you can see in the OOPS window are instances of classes with an
unique name for all instances of this class. Nevertheless the name of two instances
of different classes can be the same. */
  char* name;
  /** Red part of color */
  PyObject* R;
  /** Green part of color */
  PyObject* G;
  /** Blue part of color */
  PyObject* B;
};

#endif
```

Mesh Class Reference

[Blender Index] [Blender Hierarchy]

Blender meshes can be seen in the OOPS window as brown rectangles More...

`#include <mesh.h>`

Inherits: Blender

Public Members

- **Mesh** ()
- **~Mesh** ()
- **Mesh** (const char* name)
- PyObject* **addFace** (int i1, int i2, int i3, int i4, int isSmooth, int matIndex)
- PyObject* **addTexCoords** (float u1, float v1, float u2, float v2, float u3, float v3, float u4, float v4)
- PyObject* **addTexture** (char* filename)
- PyObject* **addVertex** (float vx, float vy, float vz, float nx, float ny, float nz, float r = -1.0, float g = 0.0, float b = 0.0)
- PyObject* **createTrianglesFromEdges** ()
- PyObject* **enterEditMode** ()
- PyObject* **leaveEditMode** ()
- PyObject* **removeDoubles** ()
- char* **name**
- PyObject* **vertices**
- PyObject* **normals**
- PyObject* **colors**
- PyObject* **faces**
- PyObject* **texcoords**
- PyObject* **texture**

Detailed Description

Blender meshes can be seen in the OOPS window as brown rectangles. They are constructed without a connection to an object but should be connected to one. This gives users the chance to "share" a mesh by connecting one mesh to different objects.

Mesh(const char* name)

A Blender mesh has allways a unique name. The name you give as an argument is only a proposal for the real name. Blender will check if this name is already used and rename the mesh if necessary.

PyObject* addFace(int i1, int i2, int i3, int i4, int isSmooth, int matIndex)

Adds a list with 6 entries to the faces list. This might change in the future. The first 4 entries are indices to describe the vertices for a triangle or a quad. The decision if a quad is used or not is made by the 4th entry. If this is 0 then only the first 3 entries are used for an triangle. The 5th entry tells if the triangle (or quad) should be rendered as "smooth" (using vertex normals) or not. The last entry is the index of the material used for this face.

PyObject* addTexCoords(float u1, float v1, float u2, float v2, float u3, float v3, float u4, float v4)

Adds the texture coordinates for one face to texcoords. The texture coordinates are stored for each face as a list of 4 sublists (for triangles just ignore the 4th sublist). Each of this sublists has 2 entries (u- and v-direction).

PyObject* addTexture(char* filename)

Replaces the filename (including path) for the texture. In the future more than only one texture might be used.

PyObject* addVertex(float vx, float vy, float vz, float nx, float ny, float nz, float r = -1.0, float g = 0.0, float b = 0.0)

Adds values for the list vertices, normals, and colors. The first 3 values are used for the vertex coordinates, the next 3 values for the vertex normal, and the last 3 (if used) for the vertex color. The

color information has not to be specified. The default value for r (-1.0) means: Don't store any color information.

PyObject* createTrianglesFromEdges()

Instead of creating triangles and quads you can also create edges with addFace(...) now. This edges can be used by calling createTrianglesFromEdges() to triangulate a general polygon (even with holes).

PyObject* enterEditMode()

This is like pressing the TAB key in Blender. You enter edit mode and you can add vertices and faces within Python. Then you leave edit mode again and the data is transfered from Python to Blender.

PyObject* leaveEditMode()

When you leave edit mode all collected data (vertices, faces) are transfered from Python to Blender.

PyObject* removeDoubles()

After importing data (e.g. a DXF file) you might have doubled vertices because the file format does not allow to reuse the same vertices by using indices. This function removes doubled vertices within a precision value (at the moment you can't use the precision value from the interface; a value of 0.0003 is used) and is also called internally by createTrianglesFromEdges().

char* name

All rectangles you can see in the OOPS window are instances of classes with an unique name for all instances of this class. Nevertheless the name of two instances of different classes can be the same.

PyObject* vertices

This holds a list of vertices for usage within Python. A face uses the indices of this vertices. This allows to store the vertices efficient and to share the same vertex between different triangles or quads.

PyObject* normals

This holds a list of vertex normals which can be used for "smooth" rendering. The number of elements in this list should be exactly the same as in the list for the vertices.

PyObject* colors

This holds a list of vertex colors. The list might be empty if no colors are used. But if they are used the list should be exactly of the same size as the list of vertices.

PyObject* faces

This holds a list of faces. At the moment a face is a list with 6 entries (integers). This might change in the future. The first 4 entries are indices to describe the vertices for a triangle or a quad. The decision if a quad is used or not is made by the 4th entry. If this is 0 then only the first 3 entries are used for an triangle. The 5th entry tells if the triangle (or quad) should be rendered as "smooth" (using vertex normals) or not. The last entry is the index of the material used for this face.

PyObject* texcoords

This holds a list of texture coordinates. The list might be empty but if used the list should have exactly the same size as the list of faces. The texture coordinates are stored for each face as a list of 4 sublists (for triangles just ignore the 4th sublist). Each of this sublists has 2 entries (u- and v- direction).

PyObject* texture

This holds the name (with path) of the used texture (if any).

- *Author*: Jan Walter **K***doc*

- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          mesh.h  -  description
                             -------------------
    begin                : Wed Dec 6 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef MESH_H
#define MESH_H

#include <blender.h>

struct PyObject;

/**Blender meshes can be seen in the OOPS window as brown rectangles. They are
constructed without a connection to an object but should be connected to one. This
gives users the chance to "share" a mesh by connecting one mesh to different objects.
  *@author Jan Walter
  */

class Mesh: public Blender {
public:
        Mesh();
        ~Mesh();
  /** A Blender mesh has allways a unique name. The name you give as an argument is
only a proposal for the real name. Blender will check if this name is already used
and rename the mesh if necessary.  */
  Mesh(const char* name);
  /** Adds a list with 6 entries to the faces list. This might change in the future.
The first 4 entries are indices to describe the vertices for a triangle or a quad.
The decision if a quad is used or not is made by the 4th entry. If this is 0 then
only the first 3 entries are used for an triangle. The 5th entry tells if the
triangle (or quad) should be rendered as "smooth" (using vertex normals) or not. The
last entry is the index of the material used for this face. */
  PyObject* addFace(int i1, int i2, int i3, int i4, int isSmooth, int matIndex);
  /** Adds the texture coordinates for one face to texcoords. The texture coordinates
are stored for each face as a list of 4 sublists (for triangles just ignore the 4th
sublist). Each of this sublists has 2 entries (u- and v- direction). */
  PyObject* addTexCoords(float u1, float v1, float u2, float v2, float u3, float v3,
float u4, float v4);
  /** Replaces the filename (including path) for the texture. In the future more than
only one texture might be used. */
  PyObject* addTexture(char* filename);
  /** Adds values for the list vertices, normals, and colors. The first 3 values are
used for the vertex coordinates, the next 3 values for the vertex normal, and the
```

```
last 3 (if used) for the vertex color. The color information has not to be specified.
The default value for r (-1.0) means: Don't store any color information. */
  PyObject* addVertex(float vx, float vy, float vz, float nx, float ny, float nz,
float r = -1.0, float g = 0.0, float b = 0.0);
  /** Instead of creating triangles and quads you can also create edges with
addFace(...) now. This edges can be used by calling createTrianglesFromEdges() to
triangulate a general polygon (even with holes). */
  PyObject* createTrianglesFromEdges();
  /** This is like pressing the TAB key in Blender. You enter edit mode and you can
add vertices and faces within Python. Then you leave edit mode again and the data is
transfered from Python to Blender. */
  PyObject* enterEditMode();
  /** When you leave edit mode all collected data (vertices, faces) are transfered
from Python to Blender. */
  PyObject* leaveEditMode();
  /** After importing data (e.g. a DXF file) you might have doubled vertices because
the file format does not allow to reuse the same vertices by using indices. This
function removes doubled vertices within a precision value (at the moment you can't
use the precision value from the interface; a value of 0.0003 is used) and is also
called internally by createTrianglesFromEdges(). */
  PyObject* removeDoubles();
private: // Private attributes
  /** All rectangles you can see in the OOPS window are instances of classes with an
unique name for all instances of this class. Nevertheless the name of two instances
of different classes can be the same. */
  char* name;
  /** This holds a list of vertices for usage within Python. A face uses the indices
of this vertices. This allows to store the vertices efficient and to share the same
vertex between different triangles or quads. */
  PyObject* vertices;
  /** This holds a list of vertex normals which can be used for "smooth" rendering.
The number of elements in this list should be exactly the same as in the list for the
vertices. */
  PyObject* normals;
  /** This holds a list of vertex colors. The list might be empty if no colors are
used. But if they are used the list should be exactly of the same size as the list of
vertices. */
  PyObject* colors;
  /** This holds a list of faces. At the moment a face is a list with 6 entries
(integers). This might change in the future. The first 4 entries are indices to
describe the vertices for a triangle or a quad. The decision if a quad is used or not
is made by the 4th entry. If this is 0 then only the first 3 entries are used for an
triangle. The 5th entry tells if the triangle (or quad) should be rendered as
"smooth" (using vertex normals) or not. The last entry is the index of the material
used for this face. */
  PyObject* faces;
  /** This holds a list of texture coordinates. The list might be empty but if used
the list should have exactly the same size as the list of faces. The texture
coordinates are stored for each face as a list of 4 sublists (for triangles just
ignore the 4th sublist). Each of this sublists has 2 entries (u- and v- direction).
*/
  PyObject* texcoords;
  /** This holds the name (with path) of the used texture (if any). */
  PyObject* texture;
};
```

```
#endif
```

Object Class Reference

[Blender Index] [Blender Hierarchy]

Blender objects can be seen in the OOPS window as gray rectangles More...

`#include <object.h>`

Inherits: Blender

Public Members

- **Object** ()
- **~Object** ()
- **Object** (const char* name)
- char* **name**
- PyObject* **matrix**
- PyObject* **inverseMatrix**
- PyObject* **materials**
- PyObject* **type**
- PyObject* **data**

Detailed Description

Blender objects can be seen in the OOPS window as gray rectangles. The have mainly a matrix and a connection to associated data like a mesh.

Object(const char* name)

A Blender object has allways a unique name. The name you give as an argument is only a proposal for the real name. Blender will check if this name is already used and rename the object if necessary.

char* name

All rectangles you can see in the OOPS window are instances of classes with an unique name for all instances of this class. Nevertheless the name of two instances of different classes can be the same.

PyObject* matrix

The matrix is stored in a PyObject for the usage of it within Python. This is a list of lists to represent a 4x4 matrix. This might change in the future.

PyObject* inverseMatrix

This holds the 4x4 inverse matrix as a list of lists for usage within Python.

PyObject* materials

The variable materials holds a list of names for usage within Python. The names can be used to get the real material settings by calling the function getMaterial(name).

PyObject* type

To know within Python which type the connected data has this variable is either None (no connected data) or a the name of a valid class. The name stored in data can be used to get the real data (e.g. if type == "Mesh": mesh = getMesh(name)).

PyObject* data

To know within Python which type the connected data has the type variable names the class and this variable is either None (no connected data) or a valid name. This name can be used to get the real data (e.g. if type == "Mesh": mesh = getMesh(name)).

- *Author*: Jan Walter

- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          object.h  -  description
                             -------------------
    begin                : Wed Dec 6 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef OBJECT_H
#define OBJECT_H

#include <blender.h>

struct PyObject;

/**Blender objects can be seen in the OOPS window as gray rectangles. The have mainly
a matrix and a connection to associated data like a mesh.
  *@author Jan Walter
  */

class Object : public Blender {
public:
        Object();
        ~Object();
  /** A Blender object has allways a unique name. The name you give as an argument is
only a proposal for the real name. Blender will check if this name is already used
and rename the object if necessary. */
  Object(const char* name);
private: // Private attributes
  /** All rectangles you can see in the OOPS window are instances of classes with an
unique name for all instances of this class. Nevertheless the name of two instances
of different classes can be the same. */
  char* name;
  /** The matrix is stored in a PyObject for the usage of it within Python. This is a
list of lists to represent a 4x4 matrix. This might change in the future. */
  PyObject* matrix;
  /** This holds the 4x4 inverse matrix as a list of lists for usage within Python.
*/
  PyObject* inverseMatrix;
  /** The variable materials holds a list of names for usage within Python. The names
can be used to get the real material settings by calling the function
getMaterial(name). */
  PyObject* materials;
  /** To know within Python which type the connected data has this variable is either
None (no connected data) or a the name of a valid class. The name stored in data can
be used to get the real data (e.g. if type == "Mesh": mesh = getMesh(name)). */
```

```
  PyObject* type;
  /** To know within Python which type the connected data has the type variable names
the class and this variable is either None (no connected data) or a valid name. This
name can be used to get the real data (e.g. if type == "Mesh": mesh = getMesh(name)).
*/
  PyObject* data;
};

#endif
```

Scene Class Reference

[Blender Index] [Blender Hierarchy]

Blender scenes can be seen in the OOPS window as green rectangles More...

`#include <scene.h>`

Inherits: Blender

Public Members

- **Scene** ()
- **~Scene** ()
- PyObject* **addObject** (PyObject* object)
- PyObject* **getCurrentCamera** ()
- char* **name**
- PyObject* **objects**

Detailed Description

Blender scenes can be seen in the OOPS window as green rectangles. A scene has a lot of objects connected to it.

PyObject* addObject(PyObject* object)

This function is used by connect(...) to make a link between an existing object and a existing scene. I guess it should not be used standalone right now.

PyObject* getCurrentCamera()

This returns the current camera for this scene.

char* name

All rectangles you can see in the OOPS window are instances of classes with an unique name for all instances of this class. Nevertheless the name of two instances of different classes can be the same.

PyObject* objects

The variable objects holds a list of names for usage within Python. The names can be used to get the real objects by calling the function getObject(name).

- *Author*: Jan Walter **K***doc*
- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          scene.h  -  description
                             ------------------
    begin                : Thu Dec 7 2000
    copyright            : (C) 2000 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef SCENE_H
#define SCENE_H

#include <blender.h>

struct PyObject;

/**Blender scenes can be seen in the OOPS window as green rectangles. A scene has a
lot of objects connected to it.
  *@author Jan Walter
  */

class Scene : public Blender {
public:
        Scene();
        ~Scene();
  /** This function is used by connect(...) to make a link between an existing object
and a existing scene. I guess it should not be used standalone right now. */
  PyObject* addObject(PyObject* object);
  /** This returns the current camera for this scene. */
  PyObject* getCurrentCamera();
private: // Private attributes
  /** All rectangles you can see in the OOPS window are instances of classes with an
unique name for all instances of this class. Nevertheless the name of two instances
of different classes can be the same. */
  char* name;
  /** The variable objects holds a list of names for usage within Python. The names
can be used to get the real objects by calling the function getObject(name). */
  PyObject* objects;
};

#endif
```

GUI Class Reference

This is in fact no class it is a module in Python More...

`#include <gui.h>`

Public Members

- **GUI** ()
- **~GUI** ()

Detailed Description

This is in fact no class it is a module in Python. But I use it here as a class to document the global functions availabe in the GUI module.

- *Author*: Jan Walter

K*doc*

- Documentation generated by jan@nvidea on Tue Feb 6 14:36:56 CET 2001

```
/***************************************************************************
                          gui.h  -  description
                             -------------------
    begin                : Fri Jan 26 2001
    copyright            : (C) 2001 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef GUI_H
#define GUI_H


/**This is in fact no class it is a module in Python. But I use it here as a class to
document the global functions availabe in the GUI module.
  *@author Jan Walter
  */

class GUI {
public:
        GUI();
        ~GUI();
};

#endif
```

FileSelector Class Reference

A file selector can be seen in Blender when you load (F1) or save (F2) a file More...

`#include <fileselector.h>`

Inherits: GUI

Public Members

- **FileSelector** ()
- **~FileSelector** ()
- PyObject* **activate** (PyObject* callback, PyObject* callbackArgs)
- PyObject* **filename**

Detailed Description

A file selector can be seen in Blender when you load (F1) or save (F2) a file. For the Python API you want to use the same mechanism to get access to the filename and the path a user can select interactively. You have to activate the FileSelector with a callback function (in Python). When the user has finished his selection the callback function is called and you can access the filename and the path in the callback function.

PyObject* activate(PyObject* callback, PyObject* callbackArgs)

With this function you start the user interaction with a file selector. The callback function is called after finishing the interaction and you can give access to the FileSelector instance by giving the callback function an (optional) argument.

PyObject* filename

This holds the file name (with path) of the selected file.

```
/***************************************************************************
                          fileselector.h  -  description
                             -------------------
    begin                : Fri Jan 26 2001
    copyright            : (C) 2001 by Jan Walter
    email                : jan@blender.nl
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifndef FILESELECTOR_H
#define FILESELECTOR_H

#include <gui.h>

struct PyObject;

/**A file selector can be seen in Blender when you load (F1) or save (F2) a file. For
the Python API you want to use the same mechanism to get access to the filename and
the path a user can select interactively. You have to activate the FileSelector with
a callback function (in Python). When the user has finished his selection the
callback function is called and you can access the filename and the path in the
callback function.
  *@author Jan Walter
  */

class FileSelector : public GUI  {
public:
        FileSelector();
        ~FileSelector();
  /** With this function you start the user interaction with a file selector. The
callback function is called after finishing the interaction and you can give access
to the FileSelector instance by giving the callback function an (optional) argument.
*/
  PyObject* activate(PyObject* callback, PyObject* callbackArgs);
private: // Private attributes
  /** This holds the file name (with path) of the selected file. */
  PyObject* filename;
};

#endif
```
