# vgio Documentation

***Release 1.3.0***

**Joshua Skelton**

**Jul 14, 2021**

# CONTENTS


**1 Installation** **3**
- 1.1 Basic Installation . . . 3
- 1.2 Install from Source . . . 3

**2 Handbook** **5**
- 2.1 Concepts . . . 5
- 2.2 Tutorial . . . 5

**3 Reference** **7**
- 3.1 devildaggers Subpackage . . . 7
- 3.2 duke3d Subpackage . . . 14
- 3.3 hexen2 Subpackage . . . 25
- 3.4 hrot Subpackage . . . 28
- 3.5 quake Subpackage . . . 30
- 3.6 quake2 Subpackage . . . 72
- 3.7 _core Subpackage . . . 96

**4 About** **101**
- 4.1 Mission . . . 101
- 4.2 License . . . 101

**5 Release Notes** **103**
- 5.1 v1.3.0 . . . 103
- 5.2 v1.2.0 . . . 103
- 5.3 v1.1.2 . . . 104
- 5.4 v1.1.1 . . . 104
- 5.5 v1.1.0 . . . 104
- 5.6 v1.0.1 . . . 104
- 5.7 v1.0.0 . . . 104

**Python Module Index** **107**

**Index** **109**

vgio is a Python library for working with data in video game file formats. Sort of like JSON but for video games.

CHAPTER

ONE

# INSTALLATION

## 1.1 Basic Installation

Install vgio with **pip**:

```
$ pip install vgio
```

## 1.2 Install from Source

Clone repo:

```
$ git clone https://github.com/joshuaskelly/vgio.git
```

Install from local sources:

```
$ cd vgio
$ pip install .
```

CHAPTER

TWO

# HANDBOOK

## 2.1 Concepts

The `vgio` library provides an API to work with data stored in various video game files formats. The library is structured such that games are represented by subpackages and file formats are represented by modules.

Broadly speaking `vgio` places file formats into three categories.

### 2.1.1 Binary Data

Binary data is structured data that is written to a file as a sequence of bytes. It is common for the data structure to be composed of other data structures.

`vgio` represents the primary binary data object of the format as a *`ReadWriteFile`* object. Helper data structures have simple `read(file)` and `write(file, object_)` classmethods.

### 2.1.2 Archive Data

Archive data is a container for other types of data typically represented as files. The *`ArchiveFile`* and *`ArchiveInfo`* serve as base classes for working with such data. By design the *`ArchiveFile`* and *`ArchiveInfo`* interfaces are identical to Python's ZipFile and ZipInfo interfaces respectively.

### 2.1.3 Markup Data

Markup data is data that is expressed as structured plain text. `vgio` modules that are markup based expose a similar interace as Python's json module interface. Namely `loads()` and `dumps()`.

## 2.2 Tutorial

### 2.2.1 Using a ReadWriteFile Class

The most common way of working with video game file formats in the `vgio` library is a class derived from the *`ReadWriteFile`* class. You can create instances of this class by loading from a file or from scratch.

To load a video game file format object from a file use the `open(file, mode='r')` classmethod on the derived *`ReadWriteFile`* class. Because *`ReadWriteFile`* is a base class this example will use the Quake *`Mdl`* model format:

```
>>> from vgio.quake.mdl import Mdl
>>> mdl_file = Mdl.open('armor.mdl')
```

If successful, it will return an *Mdl* object. You can now use instance attributes to examine the file contents:

```
>>> print(mdl_file.version)
6
>>> print(mdl_file.identifier)
b'IDPO'
```

### 2.2.2 Using an ArchiveFile Class

It is common for video games to bundle their files in an archive and the `vgio` library provides classes derived from *ArchiveFile* and *ArchiveInfo* to work with that data.

**Note:** An *ArchiveFile* object must be created using a file or file-like object.

Since the *ArchiveFile* is a base class, this example will use the Duke3D *GrpFile* archive format:

```
>>> from vgio.duke3d.grp import GrpFile
>>> grp_file = GrpFile('DUKE3D.GRP')
```

If successful, it will return an *GrpFile* object. You can now get a *GrpInfo* object and can use the instance attributes to examine the file contents:

```
>>> info = grp_file.infolist()[0]
>>> print(info.filename)
LOGO.ANM
>>> print(info.file_size)
1507336
```

CHAPTER

THREE

# REFERENCE

## 3.1 devildaggers Subpackage

**Source code:** devildaggers

### 3.1.1 hxmesh Module

**Source code:** hxmesh.py

The hxmesh module provides an *HxMesh* class which derives from *ReadWriteFile* and is used to read and write Devil Daggers mesh data.

#### HxMesh Class

**class** vgio.devildaggers.hxmesh.**HxMesh**
Class for working with HxMesh files.

##### Example

Load a file named "boid" and access mesh object attributes:

```
from vgio.devildaggers.hxmesh import HxMesh
with HxMesh.open('boid') as boid:
    indices = boid.indices
    vertices = boid.vertices
```

**indices**
An unstructured sequence of triangle indices.

**vertices**
A sequence of Vertex objects.

HxMesh.**__init__**()
Constructs an HxMesh object.

**classmethod** HxMesh.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

HxMesh.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

HxMesh.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Vertex Class

**class** vgio.devildaggers.hxmesh.**Vertex**
Class for representing an HxMesh vertex.

#### Example

Create a vertex object:

```
from vgio.devildaggers.hxmesh import Vertex
position = 0, 0, 0
normal = 0, 0, 1
uv = 0, 1
vertex = Vertex(*position, *normal, *uv)
```

**position**
Vertex position.

**normal**
Vertex normal.

**uv**
Vertex UV coordinates.

Vertex.**__init__**(*position_x*, *position_y*, *position_z*, *normal_x*, *normal_y*, *normal_z*, *u*, *v*)
Constructs an HxMesh Vertex object.

**Parameters**

- **position_x** – The position x-coordinate

- **position_y** – The position y-coordinate
- **position_z** – The position z-coordinate
- **normal_x** – The normal x-coordinate
- **normal_y** – The normal y-coordinate
- **normal_z** – The normal z-coordinate
- **u** – The UV u-coordinate
- **v** – The UV v-coordinate

### 3.1.2 hxresourcegroup Module

**Source code:** hxresourcegroup.py

The hxresourcegroup module provides an *HxResourceGroupFile* class which derives from *ArchiveFile* and is used to read and write Devil Daggers archive data.

#### HxResourceGroupFile Class

**class** vgio.devildaggers.hxresourcegroup.**HxResourceGroupFile**

Class with methods to open, read, close, and list resource group files.

##### Example

Print out file name and type of all entries in a resource group:

```
from vgio.devildaggers.hxresourcegroup import HxResourceGroupFile
with HxResourceGroupFile('dd') as resource_group:
    for info in resource_group.infolist():
        print(f'{info.filename}: {info.type}')
```

**file**

Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by HxResourceGroupFile.

**mode**

The file mode for the file-like object.

HxResourceGroupFile.**__init__**(*file*, *mode='r'*)

Open an HxResourceGroup file, where *file* can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

**Parameters**

- **file** – A path or a file-like object.
- **mode** – File mode for the file-like object.

HxResourceGroupFile.**open**(*name*, *mode='r'*)

Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

HxResourceGroupFile.**close**()
Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

HxResourceGroupFile.**read**(*name*)
Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

HxResourceGroupFile.**write**(*filename*, *arcname=None*)
Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

HxResourceGroupFile.**writestr**(*info_or_arcname*, *data*)
Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

HxResourceGroupFile.**extract**(*member*, *path=None*)
Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

HxResourceGroupFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

HxResourceGroupFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

HxResourceGroupFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

HxResourceGroupFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

### ResourceGroupInfo Class

**class** vgio.devildaggers.hxresourcegroup.**ResourceGroupInfo**
Instances of the ResourceGroupInfo class are returned by the getinfo() and infolist() methods of HxResourceGroupFile objects. Each object stores information about a single member of the HxResourceGroupFile archive.

**type**
Type of the file.

**filename**
Name of the file in the archive.

**file_offset**

**file_size**
Size of file in bytes.

**date_time**
Last modified date as Unix timestamp.

ResourceGroupInfo.**__init__**(*type_*, *filename*, *file_offset=0*, *file_size=0*, *date_time=0*)
Constructs a ResourceGroupInfo object.

**classmethod** ResourceGroupInfo.**from_file**(*filename*)
Construct an ResourceGroupInfo instance for a file on the filesystem, in preparation for adding it to an archive file. filename should be the path to a file or directory on the filesystem.

**Parameters** **filename** – A path to a file or directory.

**Returns** A ResourceGroupInfo object.

### 3.1.3 hxshader Module

**Source code:** hxshader.py

The hxshader module provides an *HxShader* class which derives from *ReadWriteFile* and is used to read and write Devil Daggers shader data.

#### HxShader Class

**class** vgio.devildaggers.hxshader.**HxShader**
Class for working with HxShaders

**name**
Shader name.

**vertex_shader**
Vertex shader code.

**fragment_shader**
Fragment shader code.

HxShader.**__init__**()
Constructs an HxShader object.

**classmethod** HxShader.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

HxShader.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

HxShader.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** `OSError` – If file argument is not a file-like object.

### 3.1.4 hxtexture Module

**Source code:** hxtexture.py

The `hxtexture` module provides an *HxTexture* class which derives from *ReadWriteFile* and is used to read and write Devil Daggers texture data.

#### HxTexture Class

**class** vgio.devildaggers.hxtexture.**HxTexture**

Class for working with HxTextures.

**texture_format**

Likely a texture format?

**width**

Texture width at mip level 0.

**height**

Texture height at mip level 0.

**mip_level_count**

Number of mip levels

**pixels**

An unstructured sequence of interleaved RGBA data as bytes.

HxTexture.**__init__**()

Constructs an HxTexture object.

**classmethod** HxTexture.**open**(*file*, *mode='r'*)

Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

HxTexture.**close**()

Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

HxTexture.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

## 3.2 duke3d Subpackage

**Source code:** duke3d

### 3.2.1 art Module

**Source code:** art.py

The art module provides an *ArtFile* class which derives from *ArchiveFile* and is used to read and write Duke3D texture data.

vgio.duke3d.art.**is_artfile**(*filename*)
Quickly see if a file is a art file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### ArtFile Class

**class** vgio.duke3d.art.**ArtFile**
Class with methods to open, read, close, and list art files.

##### Example

Basic usage:

```
from vgio.duke3d.art import ArtFile
art_file = ArtFile(file, mode='r')
```

**file**
Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by ArtFile.

**mode**
The file mode for the file-like object.

ArtFile.**__init__**(*file*, *mode='r'*)
Open an Art file, where *file* can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

**Parameters**

- **file** – A path or a file-like object.
- **mode** – File mode for the file-like object.

ArtFile.**open**(*name*, *mode='r'*)

Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

ArtFile.**close**()

Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

ArtFile.**read**(*name*)

Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

ArtFile.**write**(*filename*, *arcname=None*)

Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

ArtFile.**writestr**(*info_or_arcname*, *data*)

Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

ArtFile.**extract**(*member*, *path=None*)

Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

ArtFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

ArtFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

ArtFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

ArtFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

### ArtInfo Class

**class** vgio.duke3d.art.**ArtInfo**
Instances of the ArtInfo class are returned by the getinfo() and infolist() methods of ArtFile objects. Each object stores information about a single member of the ArtFile archive.

**tile_index**
Index of the tile in the archive.

**tile_dimensions**
The size of the tile.

**picanim**
A sequence of bitmasked tile attributes.

**file_offset**
Offset of file in bytes.

**file_size**
Size of the file in bytes.

ArtInfo.**__init__**(*tile_index*, *tile_dimensions=(0, 0)*, *file_offset=0*, *file_size=0*)
Constructs an ArtInfo object.

### 3.2.2 grp Module

**Source code:** grp.py

The grp module provides an *GrpFile* class which derives from *ArchiveFile* and is used to read and write Duke3D archive data.

vgio.duke3d.grp.**is_grpfile**(*filename*)
Quickly see if a file is a grp file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### GrpFile Class

**class** vgio.duke3d.grp.**GrpFile**
Class with methods to open, read, close, and list grp files.

##### Example

Basic usage:

```
from vgio.duke3d.grp import GrpFile
grp_file = GrpFile(file, mode='r')
```

**file**
Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by GrpFile.

**mode**
The file mode for the file-like object.

GrpFile.**__init__**(*file*, *mode='r'*)
Open an Grp file, where *file* can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

GrpFile.**open**(*name*, *mode='r'*)
Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

GrpFile.**close**()
Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

GrpFile.**read**(*name*)
Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

GrpFile.**write**(*filename*, *arcname=None*)
Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

GrpFile.**writestr**(*info_or_arcname*, *data*)
Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

GrpFile.**extract**(*member*, *path=None*)
Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

GrpFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

GrpFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

GrpFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

GrpFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

#### GrpInfo Class

**class** vgio.duke3d.grp.**GrpInfo**
Instances of the GrpInfo class are returned by the getinfo() and infolist() methods of GrpFile objects. Each object stores information about a single member of the GrpFile archive.

**filename**
Name of file.

**file_offset**
Offset of file in bytes.

**file_size**
Size of the file in bytes.

GrpInfo.**__init__**(*filename*, *file_offset=0*, *file_size=0*)
Constructs a GrpInfo object.

### 3.2.3 map Module

**Source code:** map.py

The map module provides an *Map* class which derives from *ReadWriteFile* and is used to read and write Duke3D map data.

vgio.duke3d.map.**is_mapfile**(*filename*)
Quickly see if a file is a map file by checking the magic number.

The filename argument may be a file for file-like object.

### Map Class

class vgio.duke3d.map.**Map**
Class for working with map files

#### Example

Basic usage:

```
from vgio.duke3d.map import Map
m = Map.open(file)
```

**version**
Version of the map file. Build is 7

**position_x**
Player start position x-coordinate.

**position_y**
Player start position y-coordinate.

**position_z**
Player start position z-coordinate.

**angle**
Player start angle.

**start_sector**
Sector of player start.

**sectors**
A sequence of Sector objects.

**walls**
A sequence of Wall objects.

**sprites**
A sequence of Sprite objects.

Map.**__init__**()
Constructs a Map object.

classmethod Map.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Map.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Map.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Sector Class

class vgio.duke3d.map.**Sector**(*wall_pointer*, *wall_number*, *ceiling_z*, *floor_z*, *ceiling_stat*, *floor_stat*, *ceiling_picnum*, *ceiling_heinum*, *ceiling_shade*, *ceiling_palette*, *ceiling_x_panning*, *ceiling_y_panning*, *floor_picnum*, *floor_heinum*, *floor_shade*, *floor_palette*, *floor_x_panning*, *floor_y_panning*, *visibility*, *lotag*, *hitag*, *extra*)
Class for representing a sector

**wall_pointer**
The index of the first wall.

**wall_number**
The total number of walls in sector.

**ceiling_z**
The z-coordinate of the ceiling at the first point of sector.

**floor_z**
The z-coordinate of the floor at the first point of sector.

**ceiling_stat**
A bitmasked field of properties.

**floor_stat**
A bitmasked field of properties.

**ceiling_picnum**
Texture index into Art file

**ceiling_heinum**
Slope value. 0 is parallel to the floor, 4096 is 45 degrees.

**ceiling_shade**
Shade offset for ceiling.

**ceiling_palette**
Palette lookup number. 0 is the standard palette.

**ceiling_x_panning**
Texture x align/pan value.

**ceiling_y_panning**
Texture y align/pan value.

**floor_picnum**
Texture index into Art file

**floor_heinum**
Slope value. 0 is parallel to the floor, 4096 is 45 degrees.

**floor_shade**
Shade offset for floor.

**floor_palette**
Palette lookup number. 0 is the standard palette.

**floor_x_panning**
Texture x align/pan value.

**floor_y_panning**
Texture y align/pan value.

**visibility**
Determines how fast shade changes relative to distance

**lotag**
Tag for driving behavior.

**hitag**
Tag for driving behavior.

**extra**
Tag for driving behavior.

Sector.**__init__**(*wall_pointer*, *wall_number*, *ceiling_z*, *floor_z*, *ceiling_stat*, *floor_stat*, *ceiling_picnum*, *ceiling_heinum*, *ceiling_shade*, *ceiling_palette*, *ceiling_x_panning*, *ceiling_y_panning*, *floor_picnum*, *floor_heinum*, *floor_shade*, *floor_palette*, *floor_x_panning*, *floor_y_panning*, *visibility*, *lotag*, *hitag*, *extra*)
Constructs a Sector object.

**classmethod** Sector.**read**(*file*)

**classmethod** Sector.**write**(*file*, *sector*)

### Sprite Class

**class** vgio.duke3d.map.**Sprite**(*x*, *y*, *z*, *cstat*, *picnum*, *shade*, *palette*, *clip_distance*, *x_repeat*, *y_repeat*, *x_offset*, *y_offset*, *sector_number*, *status_number*, *angle*, *owner*, *x_velocity*, *y_velocity*, *z_velocity*, *lotag*, *hitag*, *extra*)
Class for representing a sprite

**x**
X-coordinate of sprite position.

**y**
Y-coordinate of sprite position.

**z**
Z-coordinate of sprite position.

**cstat**
A bitmasked field of properties.

**shade**
Shade offset of sprite.

**palette**
Palette lookup number. 0 is the standard palette.

**clip_distance**
Size of movement clipping square.

**x_repeat**
Used to stretch texture.

**y_repeat**
Used to stretch texture.

**x_offset**
Used to center texture.

**y_offset**
Used to center texture.

**sector_number**
Current sector of sprite.

**status_number**
Current status of sprite.

**angle**
Angle the sprite is facing.

**owner**

**x_velocity**
X-coordinate of sprite velocity.

**y_velocity**
Y-coordinate of sprite velocity.

**z_velocity**
Z-coordinate of sprite velocity.

**lotag**
Tag for driving behavior.

**hitag**
Tag for driving behavior.

**extra**
Tag for driving behavior.

Sprite.**__init__**(*x*, *y*, *z*, *cstat*, *picnum*, *shade*, *palette*, *clip_distance*, *x_repeat*, *y_repeat*, *x_offset*, *y_offset*, *sector_number*, *status_number*, *angle*, *owner*, *x_velocity*, *y_velocity*, *z_velocity*, *lotag*, *hitag*, *extra*)
Constructs a Sprite object.

**classmethod** Sprite.**read**(*file*)

**classmethod** Sprite.**write**(*file*, *sprite*)

### Wall Class

class vgio.duke3d.map.**Wall**(*x*, *y*, *point2*, *next_wall*, *next_sector*, *cstat*, *picnum*, *over_picnum*, *shade*, *palette*, *x_repeat*, *y_repeat*, *x_panning*, *y_panning*, *lotag*, *hitag*, *extra*)

Class for representing a wall

**x**
X-coordinate of left side of wall.

**y**
Y-coordinate of left side of wall.

**point2**
Index to the next wall on the right.

**next_wall**
Index to wall on the other side of wall. Will be -1 if there is no sector.

**next_sector**
Index to sector on other side of wall. Will be -1 if there is no sector.

**cstat**
A bitmasked field of properties.

**picnum**
Texture index into Art file.

**over_picnum**
Texture index into Art file for masked walls.

**shade**
Shade offset of wall.

**palette**
Palette lookup number. 0 is the standard palette.

**x_repeat**
Used to stretch texture.

**y_repeat**
Used to stretch texture.

**x_panning**
Used to align/pan texture.

**y_panning**
Used to align/pan texture.

**lotag**
Tag for driving behavior.

**hitag**
Tag for driving behavior.

**extra**
Tag for driving behavior.

Wall.**__init__**(*x*, *y*, *point2*, *next_wall*, *next_sector*, *cstat*, *picnum*, *over_picnum*, *shade*, *palette*, *x_repeat*, *y_repeat*, *x_panning*, *y_panning*, *lotag*, *hitag*, *extra*)

Constructs a Wall object.

classmethod Wall.**read**(*file*)

classmethod Wall.**write**(*file*, *wall*)

vgio.duke3d.**palette**
256 color palette of RGB three-tuples.

## 3.3 hexen2 Subpackage

**Source code:** hexen2

### 3.3.1 bsp Module

**Source code:** bsp.py

The bsp module provides an *Bsp* class which derives from *Bsp* and is used to read and write Hexen 2 bsp data.

vgio.hexen2.bsp.**is_bspfile**(*filename*)
Quickly see if a file is a bsp file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### Bsp Class

**class** vgio.hexen2.bsp.**Bsp**
Class for working with Bsp files

##### Example

Basic usage:

```
from vgio.quake.bsp.bsp29a import Bsp
b = Bsp.open('ad_sepulcher.bsp')
```

**version**
Version of the map file. Vanilla Quake is 29.

**entities**
A string containing the entity definitions.

**planes**
A sequence of Planes used by the bsp tree data structure.

**miptextures**
A sequence of Miptextures.

**vertexes**
A sequence of Vertexes.

**visibilities**
A sequence of ints representing visibility data.

**nodes**
A sequence of Nodes used by the bsp tree data structure.

**texture_infos**
A sequence of TextureInfo objects.

**faces**
A sequence of Faces.

**lighting**
A sequence of ints representing lighting data.

**clip_nodes**
A sequence of ClipNodes used by the bsp tree data structure.

**leafs**
A sequence of Leafs used by the bsp tree data structure.

**mark_surfaces**
A sequence of ints representing lists of consecutive faces used by the Node objects.

**edges**
A sequence of Edges.

**surf_edges**
A sequence of ints representing list of consecutive edges used by the Face objects.

**models**
A sequence of Models.

---

**Note:** The first model is the entire level.

---

**fp**
The file-like object to read data from.

**mode**
The file mode for the file-like object.

Bsp.**__init__**()
Initializes a ReadWriteFile object. Derving classes must call this.

**classmethod** Bsp.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.

- **OSError** – If the file argument is not a file-like object.

Bsp.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Bsp.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Model Class

**class** vgio.hexen2.bsp.**Model**(*bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *origin_x*, *origin_y*, *origin_z*, *head_node_0*, *head_node_1*, *head_node_2*, *head_node_3*, *head_node_4*, *head_node_5*, *head_node_6*, *head_node_7*, *visleafs*, *first_face*, *number_of_faces*)
Class for representing a model

**bounding_box_min**
The minimum coordinate of the bounding box containing the model.

**bounding_box_max**
The maximum coordinate of the bounding box containing the model.

**origin**
The origin of the model.

**head_node**
An eight-tuple of indexes. Corresponds to number of map hulls.

**visleafs**
The number of leaves in the bsp tree?

**first_face**
The number of the first face in Bsp.mark_surfaces.

**number_of_faces**
The number of faces contained in the node. These are stored in consecutive order in Bsp.mark_surfaces starting at Model.first_face.

Model.**__init__**(*bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *origin_x*, *origin_y*, *origin_z*, *head_node_0*, *head_node_1*, *head_node_2*, *head_node_3*, *head_node_4*, *head_node_5*, *head_node_6*, *head_node_7*, *visleafs*, *first_face*, *number_of_faces*)
Constructs a Model object.

**classmethod** Model.**read**(*file*)

**classmethod** Model.**write**(*file*, *model*)

## 3.4 hrot Subpackage

**Source code:** hrot

### 3.4.1 pak Module

**Source code:** pak.py

The pak module provides an *PakFile* class which derives from *ArchiveFile* and is used to read and write Quake archive data.

vgio.hrot.pak.**is_pakfile**(*filename*)
Quickly see if a file is a pak file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### PakFile Class

**class** vgio.hrot.pak.**PakFile**
Class with methods to open, read, close, and list pak files.

##### Example

Basic usage:

```
from vgio.quake.pak import PakFile
p = PakFile(file, mode='r')
```

**Parameters**

- **file** – Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by PakFile.
- **mode** – The file mode for the file-like object.

PakFile.**__init__**(*file*, *mode='r'*)

PakFile.**open**(*name*, *mode='r'*)
Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

PakFile.**close**()
Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

PakFile.**read**(*name*)
Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

PakFile.**write**(*filename*, *arcname=None*)
Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

PakFile.**writestr**(*info_or_arcname*, *data*)
Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

PakFile.**extract**(*member*, *path=None*)
Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

PakFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.

- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

PakFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

PakFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

PakFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

#### PakInfo Class

**class** vgio.hrot.pak.**PakInfo**
Instances of the PakInfo class are returned by the getinfo() and infolist() methods of PakFile objects. Each object stores information about a single member of the PakFile archive.

**filename**
Name of file.

**file_offset**
Offset of file in bytes.

**file_size**
Size of the file in bytes.

PakInfo.**__init__**(*filename*, *file_offset=0*, *file_size=0*)

## 3.5 quake Subpackage

**Source code:** quake

### 3.5.1 bsp Subpackage

**Source code:** bsp

### bsp29 Module

**Source code:** bsp29.py

The bsp29 module provides an *Bsp* class which derives from *ReadWriteFile* and is used to read and write Quake bsp29 data.

vgio.quake.bsp.bsp29.**is_bspfile**(*filename*)
Quickly see if a file is a bsp file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### Bsp Class

**class** vgio.quake.bsp.bsp29.**Bsp**
Class for working with Bsp files

#### Example

Basic usage:

```
from vgio.quake.bsp.bsp29 import Bsp
b = Bsp.open('e1m1.bsp')
```

**version**
Version of the map file. Vanilla Quake is 29.

**entities**
A string containing the entity definitions.

**planes**
A sequence of Planes used by the bsp tree data structure.

**miptextures**
A sequence of Miptextures.

**vertexes**
A sequence of Vertexes.

**visibilities**
A sequence of ints representing visibility data.

**nodes**
A sequence of Nodes used by the bsp tree data structure.

**texture_infos**
A sequence of TextureInfo objects.

**faces**
A sequence of Faces.

**lighting**
A sequence of ints representing lighting data.

**clip_nodes**
A sequence of ClipNodes used by the bsp tree data structure.

**leafs**
A sequence of Leafs used by the bsp tree data structure.

**mark_surfaces**
A sequence of ints representing lists of consecutive faces used by the Node objects.

**edges**
A sequence of Edges.

**surf_edges**
A sequence of ints representing list of consecutive edges used by the Face objects.

**models**
A sequence of Models.

---

**Note:** The first model is the entire level.

---

**fp**
The file-like object to read data from.

**mode**
The file mode for the file-like object.

Bsp.**__init__**()
Initializes a ReadWriteFile object. Derving classes must call this.

**classmethod** Bsp.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Bsp.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Bsp.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Plane Class

class vgio.quake.bsp.bsp29.**Plane**(*normal_x*, *normal_y*, *normal_z*, *distance*, *type*)
Class for representing a bsp plane

**normal**
The normal vector to the plane.

**distance**
The distance from world (0, 0, 0) to a point on the plane

**type**
Planes are classified as follows:

0. Axial plane aligned to the x-axis.
1. Axial plane aligned to the y-axis.
2. Axial plane aligned to the z-axis.
3. Non-axial plane roughly aligned to the x-axis.
4. Non-axial plane roughly aligned to the y-axis.
5. Non-axial plane roughly aligned to the z-axis.

Plane.**__init__**(*normal_x*, *normal_y*, *normal_z*, *distance*, *type*)
Constructs a Plane object.

classmethod Plane.**read**(*file*)

classmethod Plane.**write**(*file*, *plane*)

### Miptexture Class

class vgio.quake.bsp.bsp29.**Miptexture**
Class for representing a miptexture

A miptexture is an indexed image mip map embedded within the map. Maps usually have many miptextures, and the miptexture lump is treated like a small wad file.

**name**
The name of the miptexture.

**width**
The width of the miptexture. .. note:: This is the width at mipmap level 0.

**height**
The height of the miptexture. .. note:: This is the height at mipmap level 0.

**offsets**
The offsets for each of the mipmaps. This is a tuple of size four (this is the number of mipmap levels).

**pixels**
A tuple of unstructured pixel data represented as integers. A palette must be used to obtain RGB data.

---

**Note:** This is the pixel data for all four mip levels. The size is calculated using the simplified form of the geometric series where r = 1/4 and n = 4.

---

The size of this tuple is:

miptexture.width * miptexture.height * 85 / 64

Miptexture.**__init__**()
Constructs a MipTexture object.

**classmethod** Miptexture.**read**(*file*)

**classmethod** Miptexture.**write**(*file*, *miptexture*)

### Vertex Class

**class** vgio.quake.bsp.bsp29.**Vertex**(*x*, *y*, *z*)
Class for representing a vertex

A Vertex is an XYZ triple.

**x**
The x-coordinate

**y**
The y-coordinate

**z**
The z-coordinate

Vertex.**__init__**(*x*, *y*, *z*)
Constructs a Vertex object.

**classmethod** Vertex.**read**(*file*)

**classmethod** Vertex.**write**(*file*, *vertex*)

### Node Class

**class** vgio.quake.bsp.bsp29.**Node**(*plane_number*, *child_front*, *child_back*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_face*, *number_of_faces*)
Class for representing a node

A Node is a data structure used to compose a bsp tree data structure. A child may be a Node or a Leaf.

**plane_number**
The number of the plane that partitions the node.

**children**
A two-tuple of the two sub-spaces formed by the partitioning plane.

**Note:** Child 0 is the front sub-space, and 1 is the back sub-space.

**Note:** If bit 15 is set, the child is a leaf.

**bounding_box_min**
The minimum coordinate of the bounding box containing this node and all of its children.

**bounding_box_max**
The maximum coordinate of the bounding box containing this node and all of its children.

**first_face**
The number of the first face in Bsp.mark_surfaces.

**number_of_faces**
The number of faces contained in the node. These are stored in consecutive order in Bsp.mark_surfaces starting at Node.first_face.

Node.**__init__**(*plane_number*, *child_front*, *child_back*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_face*, *number_of_faces*)
Constructs a Node object.

**classmethod** Node.**read**(*file*)

**classmethod** Node.**write**(*file*, *node*)

### TextureInfo Class

**class** vgio.quake.bsp.bsp29.**TextureInfo**(*s_x*, *s_y*, *s_z*, *s_offset*, *t_x*, *t_y*, *t_z*, *t_offset*, *miptexture_number*, *flags*)
Class for representing a texture info

**s**
The s vector in texture space represented as an XYZ three-tuple.

**s_offset**
Horizontal offset in texture space.

**t**
The t vector in texture space represented as an XYZ three-tuple.

**t_offset**
Vertical offset in texture space.

**miptexture_number**
The index of the miptexture.

**flags**
If set to 1 the texture will be animated like water.

TextureInfo.**__init__**(*s_x*, *s_y*, *s_z*, *s_offset*, *t_x*, *t_y*, *t_z*, *t_offset*, *miptexture_number*, *flags*)
Constructs a TextureInfo object

**classmethod** TextureInfo.**read**(*file*)

**classmethod** TextureInfo.**write**(*file*, *texture_info*)

### Face Class

**class** vgio.quake.bsp.bsp29.**Face**(*plane_number*, *side*, *first_edge*, *number_of_edges*, *texture_info*, *style_0*, *style_1*, *style_2*, *style_3*, *light_offset*)
Class for representing a face

**plane_number**
The plane in which the face lies.

**side**
Which side of the plane the face lies. 0 is the front, 1 is the back.

**first_edge**
The number of the first edge in Bsp.surf_edges.

**number_of_edges**
The number of edges contained within the face. These are stored in consecutive order in Bsp.surf_edges starting at Face.first_edge.

**texture_info**
The number of the texture info for this face.

**styles**
A four-tuple of lightmap styles.

**light_offset**
The offset into the lighting data.

Face.**__init__**(*plane_number*, *side*, *first_edge*, *number_of_edges*, *texture_info*, *style_0*, *style_1*, *style_2*, *style_3*, *light_offset*)
Constructs a Face object.

**classmethod** Face.**read**(*file*)

**classmethod** Face.**write**(*file*, *plane*)

### ClipNode Class

**class** vgio.quake.bsp.bsp29.**ClipNode**(*plane_number*, *child_front*, *child_back*)
Class for representing a clip node

**plane_number**
The number of the plane that partitions the node.

**children**
A two-tuple of the two sub-spaces formed by the partitioning plane.

> **Note:** Child 0 is the front sub-space, and 1 is the back sub-space.

ClipNode.**__init__**(*plane_number*, *child_front*, *child_back*)
Constructs a ClipNode object.

**classmethod** ClipNode.**read**(*file*)

**classmethod** ClipNode.**write**(*file*, *clip_node*)

### Leaf Class

**class** vgio.quake.bsp.bsp29.**Leaf**(*contents*, *visibilitiy_offset*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_mark_surface*, *number_of_marked_surfaces*, *ambient_level_0*, *ambient_level_1*, *ambient_level_2*, *ambient_level_3*)
Class for representing a leaf

**contents**
The content of the leaf. Affect the player's view.

**visibilitiy_offset**
The offset into the visibility data.

**bounding_box_min**
The minimum coordinate of the bounding box containing this node.

**bounding_box_max**
The maximum coordinate of the bounding box containing this node.

**first_mark_surface**
The number of the first face in Bsp.mark_surfaces.

**number_of_marked_surfaces**
The number of surfaces contained within the leaf. These are stored in consecutive order in Bsp.mark_surfaces starting at Leaf.first_mark_surface.

**ambient_level**
A four-tuple that represent the volume of the four ambient sounds.

Leaf.**__init__**(*contents*, *visibilitiy_offset*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_mark_surface*, *number_of_marked_surfaces*, *ambient_level_0*, *ambient_level_1*, *ambient_level_2*, *ambient_level_3*)
Constructs a Leaf object.

**classmethod** Leaf.**read**(*file*)

**classmethod** Leaf.**write**(*file*, *leaf*)

### Edge Class

**class** vgio.quake.bsp.bsp29.**Edge**(*vertex_0*, *vertex_1*)
Class for representing a edge

**vertexes**
A two-tuple of vertexes that form the edge. Vertex 0 is the start vertex, and 1 is the end vertex.

Edge.**__init__**(*vertex_0*, *vertex_1*)
Constructs an Edge object.

**classmethod** Edge.**read**(*file*)

**classmethod** Edge.**write**(*file*, *edge*)

### Model Class

**class** vgio.quake.bsp.bsp29.**Model**(*bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *origin_x*, *origin_y*, *origin_z*, *head_node_0*, *head_node_1*, *head_node_2*, *head_node_3*, *visleafs*, *first_face*, *number_of_faces*)
Class for representing a model

**bounding_box_min**
The minimum coordinate of the bounding box containing the model.

**bounding_box_max**
The maximum coordinate of the bounding box containing the model.

**origin**
The origin of the model.

**head_node**
A four-tuple of indexes. Corresponds to number of map hulls.

**visleafs**
The number of leaves in the bsp tree?

**first_face**
The number of the first face in Bsp.mark_surfaces.

**number_of_faces**
The number of faces contained in the node. These are stored in consecutive order in Bsp.mark_surfaces starting at Model.first_face.

Model.**__init__**(*bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *origin_x*, *origin_y*, *origin_z*, *head_node_0*, *head_node_1*, *head_node_2*, *head_node_3*, *visleafs*, *first_face*, *number_of_faces*)
Constructs a Model object.

**classmethod** Model.**read**(*file*)

**classmethod** Model.**write**(*file*, *model*)

### bsp29a Module

**Source code:** bsp29a.py

The bsp29a module provides an *Bsp* class which derives from *ReadWriteFile* and is used to read and write Quake bsp29a data.

vgio.quake.bsp.bsp29a.**is_bspfile**(*filename*)
Quickly see if a file is a bsp file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### Bsp Class

**class** vgio.quake.bsp.bsp29a.**Bsp**
Class for working with Bsp files

#### Example

Basic usage:

```
from vgio.quake.bsp.bsp29a import Bsp
b = Bsp.open('ad_sepulcher.bsp')
```

**version**
Version of the map file. Vanilla Quake is 29.

**entities**
A string containing the entity definitions.

**planes**
A sequence of Planes used by the bsp tree data structure.

**miptextures**
A sequence of Miptextures.

**vertexes**
A sequence of Vertexes.

**visibilities**
A sequence of ints representing visibility data.

**nodes**
A sequence of Nodes used by the bsp tree data structure.

**texture_infos**
A sequence of TextureInfo objects.

**faces**
A sequence of Faces.

**lighting**
A sequence of ints representing lighting data.

**clip_nodes**
A sequence of ClipNodes used by the bsp tree data structure.

**leafs**
A sequence of Leafs used by the bsp tree data structure.

**mark_surfaces**
A sequence of ints representing lists of consecutive faces used by the Node objects.

**edges**
A sequence of Edges.

**surf_edges**
A sequence of ints representing list of consecutive edges used by the Face objects.

**models**
A sequence of Models.

---

**Note:** The first model is the entire level.

---

**fp**
The file-like object to read data from.

**mode**
The file mode for the file-like object.

`Bsp.__init__()`
Initializes a ReadWriteFile object. Derving classes must call this.

**classmethod** `Bsp.open`(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Bsp.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Bsp.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Node Class

**class** vgio.quake.bsp.bsp29a.**Node**(*plane_number*, *child_front*, *child_back*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_face*, *number_of_faces*)

Node.**__init__**(*plane_number*, *child_front*, *child_back*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_face*, *number_of_faces*)
Constructs a Node object.

**classmethod** Node.**read**(*file*)

**classmethod** Node.**write**(*file*, *node*)

### Face Class

**class** vgio.quake.bsp.bsp29a.**Face**(*plane_number*, *side*, *first_edge*, *number_of_edges*, *texture_info*, *style_0*, *style_1*, *style_2*, *style_3*, *light_offset*)

Face.**__init__**(*plane_number*, *side*, *first_edge*, *number_of_edges*, *texture_info*, *style_0*, *style_1*, *style_2*, *style_3*, *light_offset*)
Constructs a Face object.

**classmethod** Face.**read**(*file*)

**classmethod** Face.**write**(*file*, *plane*)

### ClipNode Class

**class** vgio.quake.bsp.bsp29a.**ClipNode**(*plane_number*, *child_front*, *child_back*)

ClipNode.**__init__**(*plane_number*, *child_front*, *child_back*)
Constructs a ClipNode object.

**classmethod** ClipNode.**read**(*file*)

**classmethod** ClipNode.**write**(*file*, *clip_node*)

### Leaf Class

**class** vgio.quake.bsp.bsp29a.**Leaf**(*contents*, *visibilitiy_offset*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_mark_surface*, *number_of_marked_surfaces*, *ambient_level_0*, *ambient_level_1*, *ambient_level_2*, *ambient_level_3*)

Leaf.**__init__**(*contents*, *visibilitiy_offset*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_mark_surface*, *number_of_marked_surfaces*, *ambient_level_0*, *ambient_level_1*, *ambient_level_2*, *ambient_level_3*)
Constructs a Leaf object.

**classmethod** Leaf.**read**(*file*)

**classmethod** Leaf.**write**(*file*, *leaf*)

### Edge Class

**class** vgio.quake.bsp.bsp29a.**Edge**(*vertex_0*, *vertex_1*)

Edge.**__init__**(*vertex_0*, *vertex_1*)
Constructs an Edge object.

**classmethod** Edge.**read**(*file*)

**classmethod** Edge.**write**(*file*, *edge*)

vgio.quake.bsp.**is_bspfile**(*filename*)
Quickly see if a file is a bsp file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### Bsp Class

class vgio.quake.bsp.**Bsp**
Factory class for working with bsp files. Will automatically detect the version of the provided file and open it with the correct versioned Bsp object.

##### Example

Basic usage:

```
from vgio.quake.bsp import Bsp
b = Bsp.open('example.bsp')
```

static Bsp.**open**(*file*, *mode='r'*)
Open a Bsp object where file can be a path to a file (a string), or a file-like object.

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file.

**Returns** A Bsp object.

### 3.5.2 dem Module

**Source code:** dem.py

The dem module provides an *Dem* class which derives from *ReadWriteFile* and is used to read and write Quake demo data.

#### Dem Class

class vgio.quake.dem.**Dem**
Class for working with Dem files

##### Example

Basic usage:

```
from vgio.quake.dem import Dem
d = Dem.open(file)
```

**cd_track**
The number of the cd track to play. The track will be '-1' if no music.

**message_blocks**
A sequence of MessageBlock objects

Dem.**__init__**()
Constructs a Dem object

**classmethod** Dem.**open**(*file*, *mode='r'*)

Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Dem.**close**()

Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Dem.**save**(*file*)

Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### 3.5.3 lmp Module

**Source code:** lmp.py

The lmp module provides an *Lmp* class which derives from *ReadWriteFile* and is used to read and write Quake lmp data.

#### Lmp Class

**class** vgio.quake.lmp.**Lmp**

Class for working with Lmp files

There are three different types of lump files:

1. **2D image - The majority of the lump files are 2D images. If a lump** is a 2D image it will have width, height, and pixels attributes.
2. **Palette - The palette lump has the palette attribute which is a** list of 256 RGB tuples. This is used to map color indexes to actual RGB triples.
3. **Colormap - The colormap lump has the colormap attribute which is a** list of 16384 color indexes. It functions as a 256 x 64 table for mapping colors to different values for lighting.

### Example

Basic usage:

```
from vgio.quake.lmp import Lmp
l = Lmp.open(file)
```

**width**
(2D image lump only) The width of the lump.

**height**
(2D image lump only) The height of the lump.

**pixels**
(2D image lump only) The raw pixel data.

**palette**
(Palette lump only) A sequence of 256 RGB tuples.

**colormap**
(Color Map lump only) A sequence of 16384 color indexes.

Lmp.**__init__**()
Initializes a ReadWriteFile object. Derving classes must call this.

**classmethod** Lmp.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Lmp.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Lmp.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### 3.5.4 map Module

**Source code:** map.py

vgio.quake.map.**dumps**(*entities*)
Serialize Entity objects to a formatted string.

**Parameters** **entities** – A sequence of Entity objects.

**Returns** A formatted string containing a Map document.

vgio.quake.map.**loads**(*s*)
Deserializes string s into Entity objects

**Parameters** **s** – A string containing a Map document.

**Returns** A sequence of Entity objects.

**Raises** **ParseError** – If fails to parse given document

#### Entity Class

**class** vgio.quake.map.**Entity**
Class for representing Map Entity data

**Note:** Entity properties will be set as attributes.

**brushes**
A sequence of Brush objects.

#### Brush Class

**class** vgio.quake.map.**Brush**
Class for representing Brush data

**planes**
A sequence of Plane objects

#### Plane Class

**class** vgio.quake.map.**Plane**
Class for representing planes(faces) of a Brush.

**points**
A triple of XYZ three-tuples representing three non-collinear points contained in the plane.

**texture_name**
Name of the Miptexture

**offset**
The texture offset represented as an XY two-tuple.

**rotation**
The texture rotation angle in degrees.

**scale**
The texture scale represented as an XY two-tuple.

### 3.5.5 mdl Module

**Source code:** mdl.py

The mdl module provides an *Mdl* class which derives from *ReadWriteFile* and is used to read and write Quake mdl data.

vgio.quake.mdl.**is_mdlfile**(*filename*)
Quickly see if a file is a mdl file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### Mdl Class

**class** vgio.quake.mdl.**Mdl**
Class for working with Mdl files

##### Example

Basic usage:

```
from vgio.quake.mdl import Mdl
m = Mdl.open(file)
```

**identifier**
The magic number of the file, must be b'IDPO'

**version**
The version of the file, should be 6.

**scale**
The scale of the model. Used to correctly resize the model as the frame vertexes are packed into a (0, 0, 0) to (255, 255, 255) local space.

**origin**
The offset of the model. Used to correctly position the model.

Note: The frame vertexes are packed into a (0, 0, 0) to (255, 255, 255) local space.

**bounding_radius**
The bounding radius of the model.

**eye_position**
The eye position for the model.

**number_of_skins**
The number of skins contained inside the model.

**skin_width**
The pixel width of the skin texture.

**skin_height**
The pixel height of the skin texture.

**number_of_vertexes**
The number of vertexes for the model.

**number_of_triangles**
The number of triangles for the model.

**number_of_frames**
The number of frames for the model.

**synctype**
The synchronization type for the model. It is either SYNC or RAND.

**flags**
A bit field of entity effects.

**size**
The average size of the triangles.

**skins**
The list of Skin or SkinGroup objects. Use the type attribute to identify the object. The type is either SINGLE or GROUP.

**st_vertexes**
The list of StVertex objects.

**triangles**
The list of Triangle objects.

**frames**
The list of Frame or FrameGroup objects. Use the type attribute to identify the object. The type is either SINGLE or GROUP.

Mdl.**__init__**()
Constructs an Mdl object.

**classmethod** Mdl.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Mdl.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Mdl.**save**(*file*)
Writes data to file.

**Parameters** `file` – Either the path to the file, or a file-like object.

**Raises** `OSError` – If file argument is not a file-like object.

### `Skin` Class

`class vgio.quake.mdl.Skin`
Class for representing a skin

A skin is an indexed image embedded within the model. Models may contain more than one skin, and there may be as many skins as are there are separate parts in the model.

`type`
The SkinType for the skin. For a Skin object the type must be SINGLE

`pixels`
A tuple of unstructured indexed pixel data represented as integers. A palette must be used to obtain RGB data. The size of this tuple is:

mdl.skin_width * mdl.skin_height.

`Skin.__init__()`

`static Skin.read(`*file*, *size*`)`

`static Skin.write(`*file*, *skin*, *size*`)`

### `SkinGroup` Class

`class vgio.quake.mdl.SkinGroup`
Class for representing a skin group

A skin group is a sequence of indexed images embedded within the model.

`type`
The SkinType for the skin group. For a SkinGroup the type must be GROUP

`number_of_skins`
The number of skins contain within this SkinGroup.

`intervals`
The time intervals between skin.

`pixels`
A tuple of unstructured indexed pixel data represented as integers. A palette must be used to obtain RGB data. This size of this tuple is:

mdl.skin_width * mdl.skin_height * number_of_frames

`SkinGroup.__init__()`

`static SkinGroup.read(`*file*, *size*`)`

`static SkinGroup.write(`*file*, *skin_group*, *size*`)`

### StVertex Class

class vgio.quake.mdl.**StVertex**(*on_seam*, *s*, *t*)
Class for representing an st vertex

StVertices are similar to UV coordinates but are expressed in terms of surface space and span (0,0) to (texture_width, texture_height).

**Note:** If an StVertex lies on a seam and belongs to a back facing triangle, the s-component must be incremented by half of the skin width.

**on_seam**
Indicates if the StVertex is on a skin boundary. The value will be 0 if not on the seam and 0x20 if it does lie on the seam.

**s**
The s-coordinate on the skin.

**t**
The t-coordinate on the skin.

StVertex.**__init__**(*on_seam*, *s*, *t*)

classmethod StVertex.**read**(*file*)

classmethod StVertex.**write**(*file*, *stvertex*)

### Triangle Class

class vgio.quake.mdl.**Triangle**(*faces_front*, *vertexes_0*, *vertexes_1*, *vertexes_2*)
Class for representing a triangle

**Note:** The triangle winding direction is clockwise.

**faces_front**
Indicates if the triangle faces the front of the model, or towards the back. The value will be 0 for back-facing and 0x10 for front-facing.

**vertexes**
A triple of vertex indices.

Triangle.**__init__**(*faces_front*, *vertexes_0*, *vertexes_1*, *vertexes_2*)

classmethod Triangle.**read**(*file*)

classmethod Triangle.**write**(*file*, *triangle*)

### TriVertex Class

class vgio.quake.mdl.**TriVertex**(*x*, *y*, *z*, *light_normal_index*)
Class for representing a trivertex

A TriVertex is a set of XYZ coordinates and a light normal index.

---

**Note:** The XYZ coordinates are packed into a (0, 0, 0) to (255, 255, 255) local space. The actual position can be calculated:

position = (packed_vertex * mdl.scale) + mdl.offset

---

---

**Note:** The light normal index is an index into a set of pre-calculated normal vectors. These can be found in the anorms attribute of the quake module.

---

**x**
The x-coordinate.

**y**
The y-coordinate.

**z**
The z-coordinate.

**light_normal_index**
The index for the pre-calculated normal vector of this vertex used for lighting.

TriVertex.**__init__**(*x*, *y*, *z*, *light_normal_index*)

classmethod TriVertex.**read**(*file*)

classmethod TriVertex.**write**(*file*, *trivertex*)

### Frame Class

class vgio.quake.mdl.**Frame**
Class for representing a frame

A Frame is a set of vertexes that represent the state of the model at a single frame of animation.

---

**Note:** The TriVertices that describe the bounding box do not use their light_normal_index attribute.

---

**type**
The FrameType of the frame. For a Frame object the type must be SINGLE

**bounding_box_min**
The minimum coordinate of the bounding box containing the vertexes in this frame.

**bounding_box_max**
The maximum coordinate of the bounding box containing all the vertexes in this frame.

**name**
The name of the frame.

**vertexes**
A sequence of TriVertex objects.

Frame.__init__()

static Frame.read(*file*, *number_of_vertexes*)

static Frame.write(*file*, *frame*, *number_of_vertexes*)

#### FrameGroup Class

class vgio.quake.mdl.FrameGroup
Class for representing a frame group

**type**
The FrameType of the frame group. For a Frame object the type must be GROUP

**bounding_box_min**
The minimum coordinate of the bounding box containing the vertexes of all frames in this group.

**bounding_box_max**
The maximum coordinate of the bounding box containing the vertexes of all the frames in this group.

**number_of_frames**
The number of frames in this group.

**intervals**
A sequence of timings for each frame.

**frames**
A sequence of Frame objects.

FrameGroup.__init__()

static FrameGroup.read(*file*, *number_of_vertexes*)

static FrameGroup.write(*file*, *frame_group*, *number_of_vertexes*)

### 3.5.6 pak Module

**Source code:** pak.py

The pak module provides an *PakFile* class which derives from *ArchiveFile* and is used to read and write Quake archive data.

vgio.quake.pak.is_pakfile(*filename*)
Quickly see if a file is a pak file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### PakFile Class

class vgio.quake.pak.**PakFile**
Class with methods to open, read, close, and list pak files.

#### Example

Basic usage:

```
from vgio.quake.pak import PakFile
p = PakFile(file, mode='r')
```

**Parameters**

- **file** – Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by PakFile.
- **mode** – The file mode for the file-like object.

PakFile.**__init__**(*file*, *mode='r'*)

PakFile.**open**(*name*, *mode='r'*)
Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

PakFile.**close**()
Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

PakFile.**read**(*name*)
Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

PakFile.**write**(*filename*, *arcname=None*)
Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

PakFile.**writestr**(*info_or_arcname*, *data*)
Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

PakFile.**extract**(*member*, *path=None*)
Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

PakFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

PakFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

PakFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

PakFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

#### PakInfo Class

**class** vgio.quake.pak.**PakInfo**
Instances of the PakInfo class are returned by the getinfo() and infolist() methods of PakFile objects. Each object stores information about a single member of the PakFile archive.

**filename**
Name of file.

**file_offset**
Offset of file in bytes.

**file_size**
Size of the file in bytes.

PakInfo.**__init__**(*filename*, *file_offset=0*, *file_size=0*)

### 3.5.7 protocol Module

**Source code:** protocol.py

#### Bad Class

**class** vgio.quake.protocol.**Bad**
Class for representing a Bad message

This is an error message and should not appear.

Bad.**__init__**()

**static** Bad.**read**(*file*)

**static** Bad.**write**(*file*, *bad=None*)

#### Nop Class

**class** vgio.quake.protocol.**Nop**
Class for representing a Nop message

Nop.**__init__**()

**static** Nop.**read**(*file*)

**static** Nop.**write**(*file*, *nop=None*)

### Disconnect Class

class vgio.quake.protocol.**Disconnect**
Class for representing a Disconnect message

Disconnect from the server and end the game. Typically this the last message of a demo.

Disconnect.**__init__**()

**static** Disconnect.**read**(*file*)

**static** Disconnect.**write**(*file*, *disconnect=None*)

### UpdateStat Class

class vgio.quake.protocol.**UpdateStat**
Class for representing UpdateStat messages

Updates a player state value.

**index**
The index to update in the player state array.

**value**
The new value to set.

UpdateStat.**__init__**()

**static** UpdateStat.**read**(*file*)

**static** UpdateStat.**write**(*file*, *update_stat*)

### Version Class

class vgio.quake.protocol.**Version**
Class for representing Version messages

**protocol_version**
Protocol version of the server. Quake uses 15.

Version.**__init__**()

**static** Version.**read**(*file*)

**static** Version.**write**(*file*, *version*)

### SetView Class

class vgio.quake.protocol.**SetView**
Class for representing SetView messages

Sets the camera position to the given entity.

**entity**
The entity number

SetView.**__init__**()

**static** SetView.**read**(*file*)

**static** SetView.**write**(*file*, *set_view*)

### Sound Class

class vgio.quake.protocol.**Sound**
Class for representing Sound messages

Plays a sound on a channel at a position.

**entity**
The entity that caused the sound.

**bit_mask**
A bit field indicating what data is sent.

**volume**
Optional. The sound volume or None.

**attenuation**
Optional. The sound attenuation or None.

**channel**
The sound channel, maximum of eight.

**sound_number**
The number of the sound in the sound table.

**origin**
The position of the sound.

Sound.**__init__**()

**static** Sound.**read**(*file*)

**static** Sound.**write**(*file*, *sound*)

### Time Class

class vgio.quake.protocol.**Time**
Class for representing Time messages

A time stamp that should appear in each block of messages.

**time**
The amount of elapsed time(in seconds) since the start of the game.

Time.**__init__**()

**static** Time.**read**(*file*)

**static** Time.**write**(*file*, *time*)

### Print Class

class vgio.quake.protocol.**Print**
Class for representing Print messages

Prints text in the top left corner of the screen and console.

**text**
The text to be shown.

Print.**__init__**()

static Print.**read**(*file*)

static Print.**write**(*file*, *_print*)

### StuffText Class

class vgio.quake.protocol.**StuffText**
Class for representing StuffText messages

Text sent to the client console and ran.

**text**
The text to send to the client console.

Note: This string is terminated with the newline character.

StuffText.**__init__**()

static StuffText.**read**(*file*)

static StuffText.**write**(*file*, *stuff_text*)

### SetAngle Class

class vgio.quake.protocol.**SetAngle**
Class for representing SetAngle messages

Sets the camera's orientation.

**angles**
The new angles for the camera.

SetAngle.**__init__**()

static SetAngle.**read**(*file*)

static SetAngle.**write**(*file*, *set_angle*)

### ServerInfo Class

class vgio.quake.protocol.**ServerInfo**
Class for representing ServerInfo messages

Handles the loading of assets. Usually first message sent after a level change.

**protocol_version**
Protocol version of the server. Quake uses 15.

**max_clients**
Number of clients.

**multi**
Multiplayer flag. Set to 0 for single-player and 1 for multiplayer.

**map_name**
The name of the level.

**models**
The model table as as sequence of strings.

**sounds**
The sound table as a sequence of strings.

ServerInfo.**__init__**()

**static** ServerInfo.**read**(*file*)

**static** ServerInfo.**write**(*file*, *server_data*)

### LightStyle Class

**class** vgio.quake.protocol.**LightStyle**
Class for representing a LightStyle message

Defines the style of a light. Usually happens shortly after level change.

**style**
The light style number.

**string**
A string of arbitrary length representing the brightness of the light. The brightness is mapped to the characters 'a' to 'z', with 'a' being black and 'z' being pure white.

#### Example

# Flickering light light_style_message = LightStyle() light_style.style = 0 light_style.string = 'aaazaaza-aaaaz'

LightStyle.**__init__**()

**static** LightStyle.**read**(*file*)

**static** LightStyle.**write**(*file*, *light_style*)

### UpdateName Class

**class** vgio.quake.protocol.**UpdateName**
Class for representing UpdateName messages

Sets the player's name.

**player**
The player number to update.

**name**
The new name as a string.

UpdateName.**__init__**()

**static** UpdateName.**read**(*file*)

**static** UpdateName.**write**(*file*, *update_name*)

### UpdateFrags Class

class vgio.quake.protocol.**UpdateFrags**
Class for representing UpdateFrags messages

Sets the player's frag count.

**player**
The player to update.

**frags**
The new frag count.

UpdateFrags.**__init__**()

static UpdateFrags.**read**(*file*)

static UpdateFrags.**write**(*file*, *update_frags*)

### ClientData Class

class vgio.quake.protocol.**ClientData**
Class for representing ClientData messages

Server information about this client.

**bit_mask**
A bit field indicating what data is sent.

**view_height**
Optional. The view offset from the origin along the z-axis.

**ideal_pitch**
Optional. The calculated angle for looking up/down slopes.

**punch_angle**
Optional. A triple representing camera shake.

**velocity**
Optional. Player velocity.

**item_bit_mask**
A bit field for player inventory.

**on_ground**
Flag indicating if player is on the ground.

**in_water**
Flag indicating if player is in a water volume.

**weapon_frame**
Optional. The animation frame of the weapon.

**armor**
Optional. The current armor value.

**weapon**
Optional. The model number in the model table.

**health**
The current health value.

**active_ammo**
The amount count for the active weapon.

**ammo**
The current ammo counts as a quadruple.

**active_weapon**
The actively held weapon.

`ClientData.__init__()`

**static** `ClientData.read(`*file*`)`

**static** `ClientData.write(`*file*, *client_data*`)`

### `StopSound` Class

**class** `vgio.quake.protocol.StopSound`
Class for representing StopSound messages

Stops a playing sound.

**channel**
The channel on which the sound is playing.

**entity**
The entity that caused the sound.

`StopSound.__init__()`

**static** `StopSound.read(`*file*`)`

**static** `StopSound.write(`*file*, *stop_sound*`)`

### `UpdateColors` Class

**class** `vgio.quake.protocol.UpdateColors`
Class for representing UpdateColors messages

Sets the player's colors.

**player**
The player to update.

**colors**
The combined shirt/pant color.

`UpdateColors.__init__()`

**static** `UpdateColors.read(`*file*`)`

**static** `UpdateColors.write(`*file*, *update_colors*`)`

### Particle Class

class vgio.quake.protocol.**Particle**
Class for representing Particle messages

Creates particle effects

**origin**
The origin position of the particles.

**direction**
The velocity of the particles represented as a triple.

**count**
The number of particles.

**color**
The color index of the particle.

Particle.**__init__**()

**static** Particle.**read**(*file*)

**static** Particle.**write**(*file*, *particle*)

### Damage Class

class vgio.quake.protocol.**Damage**
Class for representing Damage messages

Damage information

**armor**
The damage amount to be deducted from player armor.

**blood**
The damage amount to be deducted from player health.

**origin**
The position of the entity that inflicted the damage.

Damage.**__init__**()

**static** Damage.**read**(*file*)

**static** Damage.**write**(*file*, *damage*)

### SpawnStatic Class

class vgio.quake.protocol.**SpawnStatic**
Class for representing SpawnStatic messages

Creates a static entity

**model_index**
The model number in the model table.

**frame**
The frame number of the model.

**color_map**
The color map used to display the model.

**skin**
The skin number of the model.

**origin**
The position of the entity.

**angles**
The orientation of the entity.

SpawnStatic.**__init__**()

**static** SpawnStatic.**read**(*file*)

**static** SpawnStatic.**write**(*file*, *spawn_static*)

### SpawnBinary Class

**class** vgio.quake.protocol.**SpawnBinary**
Class for representing SpawnBinary messages

This is a deprecated message.

SpawnBinary.**__init__**()

**static** SpawnBinary.**read**(*file*)

**static** SpawnBinary.**write**(*file*)

### SpawnBaseline Class

**class** vgio.quake.protocol.**SpawnBaseline**
Class for representing SpawnBaseline messages

Creates a dynamic entity

**entity**
The number of the entity.

**model_index**
The number of the model in the model table.

**frame**
The frame number of the model.

**color_map**
The color map used to display the model.

**skin**
The skin number of the model.

**origin**
The position of the entity.

**angles**
The orientation of the entity.

SpawnBaseline.**__init__**()

**static** SpawnBaseline.**read**(*file*)

**static** SpawnBaseline.**write**(*file*, *spawn_baseline*)

### TempEntity Class

class vgio.quake.protocol.**TempEntity**
Class for representing TempEntity messages

Creates a temporary entity. The attributes of the message depend on the type of entity being created.

**type**
The type of the temporary entity.

TempEntity.**__init__**()

static TempEntity.**read**(*file*)

static TempEntity.**write**(*file*, *temp_entity*)

### SetPause Class

class vgio.quake.protocol.**SetPause**
Class for representing SetPause messages

Sets the pause state

**paused**
The pause state. 1 for paused, 0 otherwise.

SetPause.**__init__**()

static SetPause.**read**(*file*)

static SetPause.**write**(*file*, *set_pause*)

### SignOnNum Class

class vgio.quake.protocol.**SignOnNum**
Class for representing SignOnNum messages

This message represents the client state.

**sign_on**
The client state.

SignOnNum.**__init__**()

static SignOnNum.**read**(*file*)

static SignOnNum.**write**(*file*, *sign_on_num*)

### CenterPrint Class

class vgio.quake.protocol.**CenterPrint**
Class for representing CenterPrint messages

Prints text in the center of the screen.

**text**
The text to be shown.

CenterPrint.**__init__**()

static CenterPrint.**read**(*file*)

**static** CenterPrint.**write**(*file*, *center_print*)

### KilledMonster Class

**class** vgio.quake.protocol.**KilledMonster**
Class for representing KilledMonster messages

Indicates the death of a monster.

KilledMonster.**__init__**()

**static** KilledMonster.**read**(*file*)

**static** KilledMonster.**write**(*file*, *killed_monster=None*)

### FoundSecret Class

**class** vgio.quake.protocol.**FoundSecret**
Class for representing FoundSecret messages

Indicates a secret has been found.

FoundSecret.**__init__**()

**static** FoundSecret.**read**(*file*)

**static** FoundSecret.**write**(*file*, *found_secret=None*)

### SpawnStaticSound Class

**class** vgio.quake.protocol.**SpawnStaticSound**
Class for representing SpawnStaticSound messages

Creates a static sound

**origin**
The position of the sound.

**sound_number**
The sound number in the sound table.

**volume**
The sound volume.

**attenuation**
The sound attenuation.

SpawnStaticSound.**__init__**()

**static** SpawnStaticSound.**read**(*file*)

**static** SpawnStaticSound.**write**(*file*, *spawn_static_sound*)

### Intermission Class

class vgio.quake.protocol.**Intermission**
Class for representing Intermission messages

Displays the level end screen.

Intermission.**__init__**()

**static** Intermission.**read**(*file*)

**static** Intermission.**write**(*file*, *intermission=None*)

### Finale Class

class vgio.quake.protocol.**Finale**
Class for representing Finale messages

Displays the episode end screen.

**text**
The text to show.

Finale.**__init__**()

**static** Finale.**read**(*file*)

**static** Finale.**write**(*file*, *finale*)

### CdTrack Class

class vgio.quake.protocol.**CdTrack**
Class for representing CdTrack messages

Selects the cd track

**from_track**
The start track.

**to_track**
The end track.

CdTrack.**__init__**()

**static** CdTrack.**read**(*file*)

**static** CdTrack.**write**(*file*, *cd_track*)

### SellScreen Class

class vgio.quake.protocol.**SellScreen**
Class for representing SellScreen messages

Displays the help and sell screen.

SellScreen.**__init__**()

**static** SellScreen.**read**(*file*)

**static** SellScreen.**write**(*file*, *sell_screen=None*)

### CutScene Class

class vgio.quake.protocol.**CutScene**
Class for representing CutScene messages

Displays end screen and text.

**text**
The text to be shown.

CutScene.**__init__**()

static CutScene.**read**(*file*)

static CutScene.**write**(*file*, *cut_scene*)

### UpdateEntity Class

class vgio.quake.protocol.**UpdateEntity**
Class for representing UpdateEntity messages

Updates an entity.

**bit_mask**
A bit field indicating what data is sent.

**entity**
The number of the entity.

**model_index**
The number of the model in the model table.

**frame**
The frame number of the model.

**color_map**
The color map used to display the model.

**skin**
The skin number of the model.

**effects**
A bit field indicating special effects.

**origin**
The position of the entity.

**angles**
The orientation of the entity.

UpdateEntity.**__init__**()

static UpdateEntity.**read**(*file*)

static UpdateEntity.**write**(*file*, *update_entity*)

#### MessageBlock Class

**class** vgio.quake.protocol.**MessageBlock**
Class for representing a message block

**view_angles**
The client view angles.

**messages**
A sequence of messages.

MessageBlock.**__init__**()

**static** MessageBlock.**read**(*file*)

**static** MessageBlock.**write**(*file*, *message_block*)

### 3.5.8 spr Module

**Source code:** spr.py

The spr module provides an *Spr* class which derives from *ReadWriteFile* and is used to read and write Quake spr data.

vgio.quake.spr.**is_sprfile**(*filename*)
Quickly see if a file is an spr file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### Spr Class

**class** vgio.quake.spr.**Spr**
Class for working with Spr files

##### Example

Basic usage:

```
from vgio.quake.spr import Spr
s = Spr.open(file)
```

**identity**
The magic number of the model, must be b'IDSP'

**version**
The version of the model, should be 1

**type**
Type of model. Defines how the sprite orients itself relative to the camera.

**bounding_radius**
The bounding radius of the model.

**width**
The width of the model.

**height**
The height of the model.

**number_of_frames**
The number of frames (sprites or groups).

**beam_length**
???

**sync_type**
The syncronization type for the model. It is either SYNC or RAND.

**fp**
The file-like object to read data from.

**mode**
The file mode for the file-like object.

Spr.**__init__**()
Initializes a ReadWriteFile object. Derving classes must call this.

**classmethod** Spr.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Spr.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Spr.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

#### SpriteFrame Class

class vgio.quake.spr.**SpriteFrame**
Class for representing a single sprite frame

**origin**
The offset of the model. Used to correctly position the model.

**width**
The pixel width of the sprite.

**height**
The pixel height of the sprite.

**pixels**
A tuple of unstructured indexed pixel data represented as integers. A palette must be used to obtain RGB data. The size of this tuple is:

spr_sprite_frame.width * spr_sprite_frame.skin_height.

SpriteFrame.**__init__**()

static SpriteFrame.**read**(*file*)

static SpriteFrame.**write**(*file*, *sprite_frame*)

#### SpriteGroup Class

class vgio.quake.spr.**SpriteGroup**
Class for representing a sprite group

**number_of_frames**
The number of sprite frames in this group.

**intervals**
A sequence of timings for each frame.

**frames**
A sequence of SprSpriteFrame objects.

SpriteGroup.**__init__**()

static SpriteGroup.**read**(*file*)

static SpriteGroup.**write**(*file*, *sprite_group*)

### 3.5.9 wad Module

**Source code:** wad.py

The wad module provides an *WadFile* class which derives from *ArchiveFile* and is used to read and write Quake archive data.

vgio.quake.wad.**is_wadfile**(*filename*)
Quickly see if a file is a wad file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### WadFile Class

class vgio.quake.wad.**WadFile**
Class with methods to open, read, close, and list wad files.

#### Example

Basic usage:

```
from vgio.quake.wad import WadFile
p = WadFile(file, mode='r')
```

**Parameters**

- **file** – Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by WadFile.
- **mode** – The file mode for the file-like object.

WadFile.**__init__**(*file*, *mode='r'*)

WadFile.**open**(*name*, *mode='r'*)
Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

WadFile.**close**()
Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

WadFile.**read**(*name*)
Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

WadFile.**write**(*filename*, *arcname=None*)
Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

WadFile.**writestr**(*info_or_arcname*, *data*)
Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

WadFile.**extract**(*member*, *path=None*)
Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

WadFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

WadFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

WadFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

WadFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

#### WadInfo Class

**class** vgio.quake.wad.**WadInfo**

Instances of the WadInfo class are returned by the getinfo() and infolist() methods of WadFile objects. Each object stores information about a single member of the WadFile archive.

**filename**
Name of file.

**file_offset**
Offset of file in bytes.

**file_size**
Size of the file in bytes.

**compression**
Type of compression.

**disk_size**
Size of file on disk in bytes.

**type**
Type of entry.

WadInfo.**__init__**(*filename*, *file_offset=0*, *file_size=0*)

vgio.quake.**palette**
256 color palette of RGB three-tuples.

vgio.quake.**anorms**
Table of pre-calculated normals.

## 3.6 quake2 Subpackage

**Source code:** quake2

### 3.6.1 bsp Module

**Source code:** bsp.py

The bsp module provides an *Bsp* class which derives from *ReadWriteFile* and is used to read and write Quake bsp data.

vgio.quake2.bsp.**is_bspfile**(*filename*)
Quickly see if a file is a bsp file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### Bsp Class

class vgio.quake2.bsp.**Bsp**
Class for working with Bsp files

#### Example

Basic usage:

```
from vgio.quake2.bsp import Bsp
b = Bsp.open(file)
```

**identity**
Identity of the Bsp file. Should be b'IBSP'

**version**
Version of the Bsp file. Should be 38

**entities**
A string containing the entity definitions.

**planes**
A sequence of Plane objects used by the bsp tree data structure.

**vertexes**
A sequence of Vertex objects.

**visibilities**
A sequence of integers representing visibility data.

**nodes**
A sequence of Node objects used by the bsp tree data structure.

**texture_infos**
A sequence of TextureInfo objects.

**faces**
A sequence of Face objects.

**lighting**
A sequence of ints representing lighting data.

**leafs**
A sequence of Leaf objects used by the bsp tree data structure.

**leaf_faces**
A sequence of ints representing a consecutive list of faces used by the Leaf objects.

**leaf_brushes**
A sequence of ints representing a consecutive list of edges used by the Leaf objects.

**edges**
A sequence of Edge objects.

**surf_edges**
A sequence of ints representing a consecutive list of edges used by the Face objects.

**models**
A sequence of Model objects.

**brushes**
A sequence of Brush objects.

**brush_sides**
A sequence of BrushSide objects.

**pop**
Proof of purchase? Always 256 bytes of null data if present.

**areas**
A sequence of Area objects.

**area_portals**
A sequence of AreaPortal objects.

Bsp.**__init__**()
Constructs a Bsp object.

**classmethod** Bsp.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Bsp.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Bsp.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Lump Class

class vgio.quake2.bsp.**Lump**(*offset*, *length*)
Class for representing a lump.

A lump is a section of data that typically contains a sequence of data structures.

**offset**
The offset of the lump entry from the start of the file.

**length**
The length of the lump entry.

Lump.**__init__**(*offset*, *length*)

classmethod Lump.**read**(*file*)

classmethod Lump.**write**(*file*, *lump*)

### Plane Class

class vgio.quake2.bsp.**Plane**(*normal_x*, *normal_y*, *normal_z*, *distance*, *type*)
Class for representing a bsp plane

**normal**
The normal vector to the plane.

**distance**
The distance from world (0, 0, 0) to a point on the plane

**type**
Planes are classified as follows: 0: Axial plane aligned to the x-axis. 1: Axial plane aligned to the y-axis. 2: Axial plane aligned to the z-axis. 3: Non-axial plane roughly aligned to the x-axis. 4: Non-axial plane roughly aligned to the y-axis. 5: Non-axial plane roughly aligned to the z-axis.

Plane.**__init__**(*normal_x*, *normal_y*, *normal_z*, *distance*, *type*)

classmethod Plane.**read**(*file*)

classmethod Plane.**write**(*file*, *plane*)

### Vertex Class

class vgio.quake2.bsp.**Vertex**(*x*, *y*, *z*)
Class for representing a vertex

A Vertex is an XYZ triple.

**x**
The x-coordinate

**y**
The y-coordinate

**z**
The z-coordinate

Vertex.**__init__**(*x*, *y*, *z*)

classmethod Vertex.**read**(*file*)

classmethod Vertex.**write**(*file*, *vertex*)

### Node Class

class vgio.quake2.bsp.**Node**(*plane_number*, *child_front*, *child_back*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_face*, *number_of_faces*)

Class for representing a node

A Node is a data structure used to compose a bsp tree data structure. A child may be a Node or a Leaf.

**plane_number**
The number of the plane that partitions the node.

**children**
A two-tuple of the two sub-spaces formed by the partitioning plane.

**Note:** Child 0 is the front sub-space, and 1 is the back sub-space.

**Note:** If bit 15 is set, the child is a leaf.

**bounding_box_min**
The minimum coordinate of the bounding box containing this node and all of its children.

**bounding_box_max**
The maximum coordinate of the bounding box containing this node and all of its children.

**first_face**
The number of the first face in Bsp.mark_surfaces.

**number_of_faces**
The number of faces contained in the node. These are stored in consecutive order in Bsp.mark_surfaces starting at Node.first_face.

Node.**__init__**(*plane_number*, *child_front*, *child_back*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_face*, *number_of_faces*)

classmethod Node.**read**(*file*)

classmethod Node.**write**(*file*, *node*)

### TextureInfo Class

class vgio.quake2.bsp.**TextureInfo**(*s_x*, *s_y*, *s_z*, *s_offset*, *t_x*, *t_y*, *t_z*, *t_offset*, *flags*, *value*, *texture_name*, *next_texture_info*)

Class for representing a texture info

**s**
The s vector in texture space represented as an XYZ three-tuple.

**s_offset**
Horizontal offset in texture space.

**t**
The t vector in texture space represented as an XYZ three-tuple.

**t_offset**
Vertical offset in texture space.

**flags**
A bitfield of surface behaviors.

**value**

**texture_name**
The path of the texture.

**next_texture_info**
For animated textures. Sequence will be terminated with a value of -1

TextureInfo.**__init__**(*s_x*, *s_y*, *s_z*, *s_offset*, *t_x*, *t_y*, *t_z*, *t_offset*, *flags*, *value*, *texture_name*, *next_texture_info*)

**classmethod** TextureInfo.**read**(*file*)

**classmethod** TextureInfo.**write**(*file*, *texture_info*)

### Face Class

**class** vgio.quake2.bsp.**Face**(*plane_number*, *side*, *first_edge*, *number_of_edges*, *texture_info*, *style_0*, *style_1*, *style_2*, *style_3*, *light_offset*)
Class for representing a face

**plane_number**
The plane in which the face lies.

**side**
Which side of the plane the face lies. 0 is the front, 1 is the back.

**first_edge**
The number of the first edge in Bsp.surf_edges.

**number_of_edges**
The number of edges contained within the face. These are stored in consecutive order in Bsp.surf_edges starting at Face.first_edge.

**texture_info**
The number of the texture info for this face.

**styles**
A four-tuple of lightmap styles.

**light_offset**
The offset into the lighting data.

Face.**__init__**(*plane_number*, *side*, *first_edge*, *number_of_edges*, *texture_info*, *style_0*, *style_1*, *style_2*, *style_3*, *light_offset*)

**classmethod** Face.**read**(*file*)

**classmethod** Face.**write**(*file*, *plane*)

### Leaf Class

class vgio.quake2.bsp.**Leaf**(*contents*, *cluster*, *area*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_leaf_face*, *number_of_leaf_faces*, *first_leaf_brush*, *number_of_leaf_brushes*)

Class for representing a leaf

**contents**
The content of the leaf. Affect the player's view.

**cluster**
The cluster containing this leaf. -1 for no visibility info.

**area**
The area containing this leaf.

**bounding_box_min**
The minimum coordinate of the bounding box containing this node.

**bounding_box_max**
The maximum coordinate of the bounding box containing this node.

**first_leaf_face**
The number of the first face in Bsp.faces

**number_of_leaf_faces**
The number of faces contained within the leaf. These are stored in consecutive order in Bsp.faces at Leaf.first_leaf_face.

**first_leaf_brush**
The number of the first brush in Bsp.brushes

**number_of_leaf_brushes**
The number of brushes contained within the leaf. These are stored in consecutive order in Bsp.brushes at Leaf.first_leaf_brush.

Leaf.**__init__**(*contents*, *cluster*, *area*, *bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *first_leaf_face*, *number_of_leaf_faces*, *first_leaf_brush*, *number_of_leaf_brushes*)

classmethod Leaf.**read**(*file*)

classmethod Leaf.**write**(*file*, *leaf*)

### Edge Class

class vgio.quake2.bsp.**Edge**(*vertex_0*, *vertex_1*)

Class for representing a edge

**vertexes**
A two-tuple of vertexes that form the edge. Vertex 0 is the start vertex, and 1 is the end vertex.

Edge.**__init__**(*vertex_0*, *vertex_1*)

classmethod Edge.**read**(*file*)

classmethod Edge.**write**(*file*, *edge*)

### Model Class

class vgio.quake2.bsp.**Model**(*bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *origin_x*, *origin_y*, *origin_z*, *head_node*, *first_face*, *number_of_faces*)

Class for representing a model

**bounding_box_min**
The minimum coordinate of the bounding box containing the model.

**bounding_box_max**
The maximum coordinate of the bounding box containing the model.

**origin**
The origin of the model.

**head_node**
A four-tuple of indexes. Corresponds to number of map hulls.

**visleafs**
The number of leaves in the bsp tree?

**first_face**
The number of the first face in Bsp.mark_surfaces.

**number_of_faces**
The number of faces contained in the node. These are stored in consecutive order in Bsp.mark_surfaces starting at Model.first_face.

Model.**__init__**(*bounding_box_min_x*, *bounding_box_min_y*, *bounding_box_min_z*, *bounding_box_max_x*, *bounding_box_max_y*, *bounding_box_max_z*, *origin_x*, *origin_y*, *origin_z*, *head_node*, *first_face*, *number_of_faces*)

**classmethod** Model.**read**(*file*)

**classmethod** Model.**write**(*file*, *model*)

### Brush Class

class vgio.quake2.bsp.**Brush**(*first_side*, *number_of_sides*, *contents*)

Brush.**__init__**(*first_side*, *number_of_sides*, *contents*)

**classmethod** Brush.**read**(*file*)

**classmethod** Brush.**write**(*file*, *brush*)

### BrushSide Class

class vgio.quake2.bsp.**BrushSide**(*plane_number*, *texture_info*)

BrushSide.**__init__**(*plane_number*, *texture_info*)

**classmethod** BrushSide.**read**(*file*)

**classmethod** BrushSide.**write**(*file*, *brush_side*)

#### Area Class

**class** vgio.quake2.bsp.**Area**(*number_of_area_portals*, *first_area_portal*)

Area.**__init__**(*number_of_area_portals*, *first_area_portal*)

**classmethod** Area.**read**(*file*)

**classmethod** Area.**write**(*file*, *area*)

#### AreaPortal Class

**class** vgio.quake2.bsp.**AreaPortal**(*portal_number*, *other_area*)

AreaPortal.**__init__**(*portal_number*, *other_area*)

**classmethod** AreaPortal.**read**(*file*)

**classmethod** AreaPortal.**write**(*file*, *area*)

### 3.6.2 dm2 Module

**Source code:** dm2.py

The dm2 module provides an *Dm2* class which derives from *ReadWriteFile* and is used to read and write Quake dm2 data.

#### Dm2 Class

**class** vgio.quake2.dm2.**Dm2**

Class for working with Dm2 files

##### Example

Basic usage:

```
from vgio.quake2.dm2 import Dm2
d = Dm2.open(file)
```

**message_blocks**

A sequence of MessageBlock objects

### 3.6.3 md2 Module

**Source code:** md2.py

The md2 module provides an *Md2* class which derives from *ReadWriteFile* and is used to read and write Quake md2 data.

vgio.quake2.md2.**is_md2file**(*filename*)

Quickly see if a file is a md2 file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### Md2 Class

**class** vgio.quake2.md2.**Md2**
Class for working with Md2 files

#### Example

Basic usage:

```
from vgio.quake2.md2 import Md2
m = Md2.open(file)
```

**identity**
The magic number of the file, must be b'IDP2'

**version**
The version of the file, should be 8.

**skin_width**
The pixel width of the skin texture.

**skin_height**
The pixel height of the skin texture.

**frames**
A sequence of Frame objects.

**skins**
A sequence of Skin objects.

**st_vertexes**
A sequence of StVertex objects.

**triangles**
A sequence of Triangle objects.

**gl_commands**
A sequence of GlCommand objects.

Md2.**__init__**()
Constructs an Md2 object.

**classmethod** Md2.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.

- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Md2.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Md2.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### Skin Class

**class** vgio.quake2.md2.**Skin**(*name*)

Skin.**__init__**(*name*)

**classmethod** Skin.**read**(*file*)

**classmethod** Skin.**write**(*file*, *skin*)

### TriVertex Class

**class** vgio.quake2.md2.**TriVertex**(*x*, *y*, *z*, *light_normal_index*)
Class for representing a trivertex

A TriVertex is a set of XYZ coordinates and a light normal index.

---

**Note:** The XYZ coordinates are packed into a (0, 0, 0) to (255, 255, 255) local space. The actual position can be calculated:

position = (packed_vertex * frame.scale) + frame.translate

---

---

**Note:** The light normal index is an index into a set of pre-calculated normal vectors. These can be found in the anorms attribute of the quake2 module.

---

**x**
The x-coordinate

**y**
The y-coordinate

**z**
The z-coordinate

**light_normal_index**
The index for the pre-calculated normal vector of this vertex used for lighting.

TriVertex.**__init__**(*x*, *y*, *z*, *light_normal_index*)

**classmethod** TriVertex.**read**(*file*)

**classmethod** TriVertex.**write**(*file*, *tri_vertex*)

### StVertex Class

**class** vgio.quake2.md2.**StVertex**(*s*, *t*)

Class for representing an st vertex

StVertices are similar to UV coordinates but are expressed in terms of surface space and span (0,0) to (texture_width, texture_height).

**Note:** If an StVertex lies on a seam and belongs to a back facing triangle, the s-component must be incremented by half of the skin width.

**s**

The x-coordinate on the skin.

**t**

The y-coordinate on the skin.

StVertex.**__init__**(*s*, *t*)

**classmethod** StVertex.**read**(*file*)

**classmethod** StVertex.**write**(*file*, *st_vertex*)

### Triangle Class

**class** vgio.quake2.md2.**Triangle**(*vertex_0*, *vertex_1*, *vertex_2*, *st_vertex_0*, *st_vertex_1*, *st_vertex_2*)

Class for representing a triangle

**Note:** The triangle winding direction is clockwise.

**vertexes**

A triple of vertex indexes. XYZ data can be obtained by indexing into the frame.vertexes attribute.

Triangle.**__init__**(*vertex_0*, *vertex_1*, *vertex_2*, *st_vertex_0*, *st_vertex_1*, *st_vertex_2*)

**classmethod** Triangle.**read**(*file*)

**classmethod** Triangle.**write**(*file*, *triangle*)

#### Frame Class

class vgio.quake2.md2.**Frame**(*scale_x*, *scale_y*, *scale_z*, *translate_x*, *translate_y*, *translate_z*, *name*)
Class for representing a frame

A Frame is an object that represents the state of the model at a single frame of animation.

**scale**
The frame scale

**translate**
The frame offset

**name**
The name of the frame.

**vertexes**
A sequence of TriVertex objects.

Frame.**__init__**(*scale_x*, *scale_y*, *scale_z*, *translate_x*, *translate_y*, *translate_z*, *name*)

**classmethod** Frame.**read**(*file*, *number_of_vertexes*)

**classmethod** Frame.**write**(*file*, *frame*)

#### GlVertex Class

**class** vgio.quake2.md2.**GlVertex**(*s*, *t*, *vertex*)

GlVertex.**__init__**(*s*, *t*, *vertex*)

**classmethod** GlVertex.**read**(*file*)

**classmethod** GlVertex.**write**(*file*, *gl_vertex*)

#### GlCommand Class

**class** vgio.quake2.md2.**GlCommand**(*mode*)

GlCommand.**__init__**(*mode*)

**classmethod** GlCommand.**read**(*file*)

**classmethod** GlCommand.**write**(*file*, *gl_command*)

### 3.6.4 pak Module

**Source code:** pak.py

The pak module provides an *PakFile* class which derives from *ArchiveFile* and is used to read and write Quake archive data.

vgio.quake2.pak.**is_pakfile**(*filename*)
Quickly see if a file is a pak file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

### PakFile Class

class vgio.quake2.pak.**PakFile**
Class with methods to open, read, close, and list pak files.

#### Example

Basic usage:

```
from vgio.quake2.pak import PakFile
p = PakFile(file, mode='r')
```

**Parameters**

- **file** – Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by PakFile.
- **mode** – The file mode for the file-like object.

PakFile.**__init__**(*file*, *mode='r'*)

PakFile.**open**(*name*, *mode='r'*)
Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

PakFile.**close**()
Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

PakFile.**read**(*name*)
Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

PakFile.**write**(*filename*, *arcname=None*)
Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

PakFile.**writestr**(*info_or_arcname*, *data*)
Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

PakFile.**extract**(*member*, *path=None*)
Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

PakFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

PakFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

PakFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

PakFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

#### PakInfo Class

**class** vgio.quake2.pak.**PakInfo**
Class with attributes describing each entry in the pak file archive.

PakInfo.**__init__**(*filename*, *file_offset=0*, *file_size=0*)

### 3.6.5 protocol Module

**Source code:** protocol.py

#### Bad Class

**class** vgio.quake2.protocol.**Bad**
Class for representing a Bad message

This is an error message and should not appear.

Bad.**__init__**()

**classmethod** Bad.**read**(*file*)

**classmethod** Bad.**write**(*file*, *bad=None*)

#### MuzzleFlash Class

**class** vgio.quake2.protocol.**MuzzleFlash**(*entity*, *weapon*)
Class for representing a MuzzleFlash message

Muzzle flashes for player weapons.

**entity**
The entity number

**weapon**
The weapon id

MuzzleFlash.**__init__**(*entity*, *weapon*)

**classmethod** MuzzleFlash.**read**(*file*)

**classmethod** MuzzleFlash.**write**(*file*, *muzzle_flash*)

#### MuzzleFlash2 Class

**class** vgio.quake2.protocol.**MuzzleFlash2**(*entity*, *flash_number*)
Class for representing a MuzzleFlash2 message

Muzzle flashes for enemy weapons.

**entity**
The entity number

**flash_number**
The flash number

MuzzleFlash2.__init__(*entity*, *flash_number*)

classmethod MuzzleFlash2.read(*file*)

classmethod MuzzleFlash2.write(*file*, *muzzle_flash2*)

### TempEntity Class

class vgio.quake2.protocol.TempEntity(*type*, **args*)
Class for representing a Temp_entity message

Creates a temporary entity. The attributes of the message depend on the type of entity being created.

**type**
The type of the temporary entity.

TempEntity.__init__(*type*, **args*)

classmethod TempEntity.read(*file*)

classmethod TempEntity.write(*file*, *temp_entity*)

### Layout Class

class vgio.quake2.protocol.Layout(*text*)
Class for representing a Layout message

Updates the player's field computer via a simple scripting language.

**text**
Script source as plain text

Layout.__init__(*text*)

classmethod Layout.read(*file*)

classmethod Layout.write(*file*, *layout*)

### Inventory Class

class vgio.quake2.protocol.Inventory(*inventory*)
Class for representing a Inventory message

**inventory**
A sequence of exactly 256 integers representing the player's current inventory item counts.

Inventory.__init__(*inventory*)

classmethod Inventory.read(*file*)

classmethod Inventory.write(*file*, *inventory*)

### Nop Class

class vgio.quake2.protocol.**Nop**
Class for representing a Nop message

Nop.**__init__**()

classmethod Nop.**read**(*file*)

classmethod Nop.**write**(*file*, *nop=None*)

### Disconnect Class

class vgio.quake2.protocol.**Disconnect**
Class for representing a Disconnect message

Disconnect.**__init__**()

classmethod Disconnect.**read**(*file*)

classmethod Disconnect.**write**(*file*, *disconnect=None*)

### Reconnect Class

class vgio.quake2.protocol.**Reconnect**
Class for representing a Reconnect message

Reconnect.**__init__**()

classmethod Reconnect.**read**(*file*)

classmethod Reconnect.**write**(*file*, *reconnect=None*)

### Sound Class

class vgio.quake2.protocol.**Sound**(*flags*, *sound_number*, *volume=1.0*, *attenuation=1*, *offset=0*, *channel=0*, *entity=0*, *position=None*)
Class for representing a Sound message

**flags**
A bit field indicating what data is sent.

**sound_number**
The sound number

**volume**
The sound volume

**attenuation**
The sound attenuation

**offset**
The offset between the frame start and sound start

**channel**
The sound channel, maximum of eight.

**entity**
The entity that owns the sound

**position**
The position of the sound

Sound.**__init__**(*flags*, *sound_number*, *volume=1.0*, *attenuation=1*, *offset=0*, *channel=0*, *entity=0*, *position=None*)

**classmethod** Sound.**read**(*file*)

**classmethod** Sound.**write**(*file*, *sound*)

### Print Class

**class** vgio.quake2.protocol.**Print**(*level*, *text*)
Class for representing a Print message

**level**
Priority level of print

**text**
The print text

Print.**__init__**(*level*, *text*)

**classmethod** Print.**read**(*file*)

**classmethod** Print.**write**(*file*, *print*)

### StuffText Class

**class** vgio.quake2.protocol.**StuffText**(*text*)
Class for representing a StuffText message

**text**
The text sent to the client console.

StuffText.**__init__**(*text*)

**classmethod** StuffText.**read**(*file*)

**classmethod** StuffText.**write**(*file*, *stuff_text*)

### ServerData Class

**class** vgio.quake2.protocol.**ServerData**(*protocol_version*, *server_count*, *attract_loop*, *game_directory*, *player_number*, *map_name*)
Class for representing a ServerData message

**protocol_version**
Protocol version of the server.

**server_count**
Server identification.

**attract_loop**
The demo type. A value of 0 indicates over wire network data.

**game_directory**
The game directory. The default is the empty string which indicates 'baseq2'

**player_number**
The player id.

**map_name**
The name of the level.

ServerData.**__init__**(*protocol_version*, *server_count*, *attract_loop*, *game_directory*, *player_number*, *map_name*)

**classmethod** ServerData.**read**(*file*)

**classmethod** ServerData.**write**(*file*, *server_data*)

### ConfigString Class

**class** vgio.quake2.protocol.**ConfigString**(*index*, *text*)
Class for representing a ConfigString message

ConfigString.**__init__**(*index*, *text*)

**classmethod** ConfigString.**read**(*file*)

**classmethod** ConfigString.**write**(*file*, *config_string*)

### SpawnBaseline Class

**class** vgio.quake2.protocol.**SpawnBaseline**(*number=0*, *model_index=0*, *model_index_2=0*, *model_index_3=0*, *model_index_4=0*, *frame=0*, *skin_number=0*, *effects=0*, *render_fx=0*, *origin_x=0*, *origin_y=0*, *origin_z=0*, *angles_x=0*, *angles_y=0*, *angles_z=0*, *old_origin_x=0*, *old_origin_y=0*, *old_origin_z=0*, *sound=0*, *event=0*, *solid=0*)
Class for representing a SpawnBaseline message

https://github.com/id-Software/Quake-2/blob/372afde46e7defc9dd2d719a1732b8ace1fa096e/client/cl_parse.c#L356

SpawnBaseline.**__init__**(*number=0*, *model_index=0*, *model_index_2=0*, *model_index_3=0*, *model_index_4=0*, *frame=0*, *skin_number=0*, *effects=0*, *render_fx=0*, *origin_x=0*, *origin_y=0*, *origin_z=0*, *angles_x=0*, *angles_y=0*, *angles_z=0*, *old_origin_x=0*, *old_origin_y=0*, *old_origin_z=0*, *sound=0*, *event=0*, *solid=0*)

**classmethod** SpawnBaseline.**read**(*file*)

**classmethod** SpawnBaseline.**write**(*file*, *spawn_baseline*)

### CenterPrint Class

**class** vgio.quake2.protocol.**CenterPrint**(*text=''*)
Class for representing a Centerprint message

CenterPrint.**__init__**(*text=''*)

**classmethod** CenterPrint.**read**(*file*)

**classmethod** CenterPrint.**write**(*file*, *center_print*)

### Download Class

class vgio.quake2.protocol.**Download**
Class for representing a Download message

https://github.com/id-Software/Quake-2/blob/372afde46e7defc9dd2d719a1732b8ace1fa096e/client/cl_parse.c#L195

Download.**__init__**()

classmethod Download.**read**(*file*)

classmethod Download.**write**(*file*, *download*)

### PlayerInfo Class

class vgio.quake2.protocol.**PlayerInfo**
Class for representing a PlayerInfo message

PlayerInfo.**__init__**()

classmethod PlayerInfo.**read**(*file*)

classmethod PlayerInfo.**write**(*file*, *player_info*)

### PacketEntities Class

class vgio.quake2.protocol.**PacketEntities**
Class for representing a PacketEntities message

PacketEntities.**__init__**()

classmethod PacketEntities.**read**(*file*)

classmethod PacketEntities.**write**(*file*, *packet_entities*)

### DeltaPacketEntities Class

class vgio.quake2.protocol.**DeltaPacketEntities**
Class for representing a DeltaPacketEntities message

DeltaPacketEntities.**__init__**()

classmethod DeltaPacketEntities.**read**(*file*)

classmethod DeltaPacketEntities.**write**(*file*, *delta_packet_entities*)

### Frame Class

class vgio.quake2.protocol.**Frame**(*server_frame=0*, *delta_frame=0*, *areas=()*)
Class for representing a Frame message

**server_frame**

**delta_frame**

**areas**

Frame.**__init__**(*server_frame=0*, *delta_frame=0*, *areas=()*)

**classmethod** Frame.**read**(*file*)

**classmethod** Frame.**write**(*file*, *frame*)

#### MessageBlock Class

**class** vgio.quake2.protocol.**MessageBlock**
Class for representing a message block

**messages**
A sequence of messages.

MessageBlock.**__init__**()

**static** MessageBlock.**read**(*file*)

**static** MessageBlock.**write**(*file*, *message_block*)

### 3.6.6 sp2 Module

**Source code:** sp2.py

The sp2 module provides an *Sp2* class which derives from *ReadWriteFile* and is used to read and write Quake sp2 data.

vgio.quake2.sp2.**is_sp2file**(*filename*)
Quickly see if a file is a sp2 file by checking the magic number.

The filename argument may be a file for file-like object.

**Parameters** **filename** – File to check as string or file-like object.

**Returns** True if given file's magic number is correct.

#### Sp2 Class

**class** vgio.quake2.sp2.**Sp2**
Class for working with Sp2 files

##### Example

Basic usage:

```
from vgio.quake2.sp2 import Sp2
s = sp2.Sp2.open(file)
```

**identity**
The identity of the file. Should be b'IDS2'

**version**
The version of the file. Should be 2.

**number_of_frames**
The number of sprite frames.

**frames**
A sequence of SpriteFrame objects.

Sp2.__init__()
Constructs a Sp2 object.

classmethod Sp2.open(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Sp2.close()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Sp2.save(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### SpriteFrame Class

class vgio.quake2.sp2.SpriteFrame(*width*, *height*, *origin_x*, *origin_y*, *name*)
Class for working with sprite frames

**width**
Width of the frame.

**height**
Height of the frame.

**origin**
The offset of the frame.

**name**
The name of the pcx file to use for the frame.

SpriteFrame.__init__(*width*, *height*, *origin_x*, *origin_y*, *name*)
Constructs a SpriteFrame object.

### 3.6.7 wal Module

**Source code:** wal.py

The wal module provides an *Wal* class which derives from *ReadWriteFile* and is used to read and write Quake wal data.

#### Wal Class

**class** vgio.quake2.wal.**Wal**
Class for working with Wal files

##### Example

Basic usage:

```
from vgio.quake2.wal import Wal
with open(path) as file:
    w = Wal.read(file)
```

**name**
The name of the wal texture.

**width**
The width of the wal texture.

**Note:** This is the width at mipmap level 0.

**height**
The height of the wal texture.

**Note:** This is the height at mipmap level 0.

**offsets**
The offsets for each of the mipmaps. This is a tuple of size four (this is the number of mipmap levels).

**animation_name**
The name of the next wal texture in the animation sequence.

**flags**
A bitfield of surface behaviors.

**contents**

**value**

**pixels**
A bytes object of unstructured indexed color data. A palette must be used to obtain RGB data.

**Note:** This is the pixel data for all four mip levels. The size is calculated using the simplified form of the geometric series where r = 1/4 and n = 4.

The size of this tuple is:

wal.width * wal.height * 85 / 64

Wal.**__init__**()
Initializes a ReadWriteFile object. Derving classes must call this.

**classmethod** Wal.**open**(*file*, *mode='r'*)
Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

Wal.**close**()
Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

Wal.**save**(*file*)
Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

vgio.quake2.**anorms**
Table of pre-calculated normals.

## 3.7 _core Subpackage

**Source code:** _core

The _core module provides several base classes for creating binary serialization classes.

**Note:** These classes must be subclassed as the don't do anything useful on their own.

### 3.7.1 ReadWriteFile Class

**class** vgio._core.**ReadWriteFile**

ReadWriteFile serves as base class for serializing/deserialing binary data.

**fp**

The handle to the open file. Will be None if file closed.

**mode**

File mode. Is one of 'r', 'w', or 'a'.

**classmethod** ReadWriteFile.**open**(*file*, *mode='r'*)

Open a ReadWriteFile object where file can be a path to a file (a string), or a file-like object.

The mode parameter should be 'r' to read an existing file, 'w' to truncate and write a new file, or 'a' to append to an existing file.

open() is also a context manager and supports the with statement:

```
with ReadWriteFile.open('file.ext') as file:
    file.save('file2.ext')
```

**Parameters**

- **file** – Either the path to the file, a file-like object, or bytes.
- **mode** – An optional string that indicates which mode to open the file

**Returns** An ReadWriteFile object constructed from the information read from the file-like object.

**Raises**

- **ValueError** – If an invalid file mode is given.
- **TypeError** – If attempting to write to a bytes object.
- **OSError** – If the file argument is not a file-like object.

ReadWriteFile.**close**()

Closes the file pointer if possible. If mode is 'w' or 'a', the file will be written to.

ReadWriteFile.**save**(*file*)

Writes data to file.

**Parameters** **file** – Either the path to the file, or a file-like object.

**Raises** **OSError** – If file argument is not a file-like object.

### 3.7.2 ArchiveFile Class

**class** vgio._core.**ArchiveFile**(*file*, *mode='r'*)

ArchiveFile serves as base class for working with binary archive data.

**file**

Either the path to the file, or a file-like object. If it is a path, the file will be opened and closed by ArchiveFile.

**mode**

File mode. Is one of 'r', 'w', or 'a'.

ArchiveFile.**open**(*name*, *mode='r'*)

Access a member of the archive as a binary file-like object. name can be either the name of a file within the archive or an ArchiveInfo object. The mode parameter, if included, must be 'r' (the default) or 'w'.

open() is also a context manager and supports the with statement:

```
with ArchiveFile('archive.file') as archive_file:
    with archive_file.open('entry') as entry_file:
        print(entry_file.read())
```

**Parameters**

- **name** – Name or ArchiveInfo object.
- **mode** – File mode to open object.

**Returns** A binary file-like object.

**Raises**

- **ValueError** – If mode isn't 'r' or 'w'.
- **RuntimeError** – If file was already closed.

ArchiveFile.**close**()

Close the archive file. You must call close() before exiting your program or essential records will not be written.

**Raises** **ValueError** – If open writing handles exist.

ArchiveFile.**read**(*name*)

Return the bytes of the file name in the archive. name is the name of the file in the archive, or a ArchiveInfo object. The archive must be open for read or append.

**Parameters** **name** – ArchiveInfo name.

**Returns** File as bytes.

ArchiveFile.**write**(*filename*, *arcname=None*)

Write the file named filename to the archive, giving it the archive name arcname (by default, this will be the same as filename, but without a drive letter and with leading path separators removed). The archive must be open with mode 'w', or 'a'.

**Parameters**

- **filename** –
- **arcname** – Optional. Name of the info object. If omitted filename will be used.

ArchiveFile.**writestr**(*info_or_arcname*, *data*)

Write a file into the archive. The contents is data, which may be either a string or a bytes instance; if it is a string, it is encoded as UTF-8 first. info_or_arcname is either the file name it will be given in the archive, or a ArchiveInfo instance. If it's an instance, at least the filename must be given. The archive must be opened with mode 'w' or 'a'.

**Parameters**

- **info_or_arcname** –
- **data** – Data to be written. Either a string or bytes.

ArchiveFile.**extract**(*member*, *path=None*)

Extract a member from the archive to the current working directory; member must be its full name or a ArchiveInfo object. Its file information is extracted as accurately as possible. path specifies a different directory to extract to. member can be a filename or an ArchiveInfo object.

**Parameters**

- **member** – Either the name of the member to extract or a ArchiveInfo instance.
- **path** – The directory to extract to. The current working directory will be used if None.

**Returns** Path to extracted file.

ArchiveFile.**extractall**(*path=None*, *members=None*)
Extract all members from the archive to the current working directory. path specifies a different directory to extract to. members is optional and must be a subset of the list returned by namelist().

**Parameters**

- **path** – The directory to extract to. The current working directory will be used if None.
- **members** – The names of the members to extract. This must be a subset of the list returned by namelist(). All members will be extracted if None.

ArchiveFile.**getinfo**(*name*)
Return a ArchiveInfo object with information about the archive member name. Calling getinfo() for a name not currently contained in the archive will raise a KeyError.

**Parameters** **name** – AchiveInfo name.

**Returns** An ArchiveInfo object.

**Raises** **KeyError** – If no archive item exists for the given name.

ArchiveFile.**infolist**()
Return a list containing an ArchiveInfo object for each member of the archive. The objects are in the same order as their entries in the actual archive file on disk if an existing archive was opened.

**Returns** A sequence of ArchiveInfo objects.

ArchiveFile.**namelist**()
Return a list of archive members by name.

**Returns** A sequence of filenames.

### 3.7.3 ArchiveInfo Class

**class** vgio._core.**ArchiveInfo**(*filename*, *file_offset=0*, *file_size=0*)
ArchiveInfo objects store information about a single entry in the ArchiveFile archive. Instances of the ArchiveInfo class are returned by the getinfo() and infolist() methods of ArchiveFile objects.

**filename**
Name of file.

**file_offset**
Offset of file in bytes.

**file_size**
Size of the file in bytes.

**classmethod** ArchiveInfo.**from_file**(*filename*)
Construct an ArchiveInfo instance for a file on the filesystem, in preparation for adding it to an archive file. filename should be the path to a file or directory on the filesystem.

**Parameters** **filename** – A path to a file or directory.

**Returns** An ArchiveInfo object.

CHAPTER

FOUR

# ABOUT

## 4.1 Mission

- *Pythonic*: Clean and well written Python.
- *Domain*-specific: The APIs and objects reflect the source code and community knowledge.
- *Complete*: Support as many file types as possible.
- *Robust*: The APIs and objects are thoroughly unit tested.

## 4.2 License

CHAPTER

FIVE

# RELEASE NOTES

## 5.1 v1.3.0

### 5.1.1 API Additions

#### Hexen 2 Subpackage

vgio now supports the BSP file format for Hexen 2.

### 5.1.2 Bug Fixes

#### _core Subpackage

- Improved support for working with ArchiveFile file objects.

#### quake Subpackage

- Fixed issue with using bsp subpackge `open()` method to load bsp files.
- Fixed issue with parsing floats of form 1.50e-08

## 5.2 v1.2.0

### 5.2.1 API Additions

#### hrot Subpackage

vgio now supports the PAK file format for HROT.

## 5.3 v1.1.2

### 5.3.1 Bug Fixes

#### quake2 Subpackage

- Fixes broken md2 Skin serialization.

## 5.4 v1.1.1

### 5.4.1 Bug Fixes

#### quake Subpackage

- Fixes broken bsp29 SurfEdge serialization.

## 5.5 v1.1.0

### 5.5.1 API Additions

#### devildaggers Subpackage

vgio now supports file formats for Devil Daggers.

### 5.5.2 API Changes

#### _core Subpackage

The _core subpackage provides base classes for common class patterns.

## 5.6 v1.0.1

## 5.7 v1.0.0

### 5.7.1 API Additions

#### duke3d Subpackage

vgio now supports file formats for Duke3D.

### quake Subpackage

vgio now supports file formats for Quake.

### quake2 Subpackage

vgio now supports file formats for Quake2.

# PYTHON MODULE INDEX

## V


vgio._core, 96
vgio._core.__init__, 96
vgio.devildaggers.hxmesh, 7
vgio.devildaggers.hxresourcegroup, 9
vgio.devildaggers.hxshader, 12
vgio.devildaggers.hxtexture, 13
vgio.duke3d, 14
vgio.duke3d.art, 14
vgio.duke3d.grp, 16
vgio.duke3d.map, 19
vgio.hexen2, 25
vgio.hexen2.bsp, 25
vgio.hrot, 27
vgio.hrot.pak, 28
vgio.quake, 30
vgio.quake.bsp, 30
vgio.quake.bsp.bsp29, 30
vgio.quake.bsp.bsp29a, 38
vgio.quake.lmp, 43
vgio.quake.mdl, 45
vgio.quake.pak, 51
vgio.quake.protocol, 54
vgio.quake.spr, 67
vgio.quake.wad, 69
vgio.quake2, 72
vgio.quake2.bsp, 72
vgio.quake2.dm2, 80
vgio.quake2.md2, 80
vgio.quake2.pak, 84
vgio.quake2.protocol, 87
vgio.quake2.sp2, 93
vgio.quake2.wal, 94

# INDEX

## Symbols


__init__() (*vgio.devildaggers.hxmesh.HxMesh method*), 7
__init__() (*vgio.devildaggers.hxmesh.Vertex method*), 8
__init__() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 9
__init__() (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo method*), 11
__init__() (*vgio.devildaggers.hxshader.HxShader method*), 12
__init__() (*vgio.devildaggers.hxtexture.HxTexture method*), 13
__init__() (*vgio.duke3d.art.ArtFile method*), 14
__init__() (*vgio.duke3d.art.ArtInfo method*), 16
__init__() (*vgio.duke3d.grp.GrpFile method*), 17
__init__() (*vgio.duke3d.grp.GrpInfo method*), 19
__init__() (*vgio.duke3d.map.Map method*), 20
__init__() (*vgio.duke3d.map.Sector method*), 22
__init__() (*vgio.duke3d.map.Sprite method*), 23
__init__() (*vgio.duke3d.map.Wall method*), 24
__init__() (*vgio.hexen2.bsp.Bsp method*), 26
__init__() (*vgio.hexen2.bsp.Model method*), 27
__init__() (*vgio.hrot.pak.PakFile method*), 28
__init__() (*vgio.hrot.pak.PakInfo method*), 30
__init__() (*vgio.quake.bsp.bsp29.Bsp method*), 32
__init__() (*vgio.quake.bsp.bsp29.ClipNode method*), 36
__init__() (*vgio.quake.bsp.bsp29.Edge method*), 37
__init__() (*vgio.quake.bsp.bsp29.Face method*), 36
__init__() (*vgio.quake.bsp.bsp29.Leaf method*), 37
__init__() (*vgio.quake.bsp.bsp29.Miptexture method*), 34
__init__() (*vgio.quake.bsp.bsp29.Model method*), 38
__init__() (*vgio.quake.bsp.bsp29.Node method*), 35
__init__() (*vgio.quake.bsp.bsp29.Plane method*), 33
__init__() (*vgio.quake.bsp.bsp29.TextureInfo method*), 35
__init__() (*vgio.quake.bsp.bsp29.Vertex method*), 34
__init__() (*vgio.quake.bsp.bsp29a.Bsp method*), 39
__init__() (*vgio.quake.bsp.bsp29a.ClipNode method*), 41
__init__() (*vgio.quake.bsp.bsp29a.Edge method*), 41
__init__() (*vgio.quake.bsp.bsp29a.Face method*), 40
__init__() (*vgio.quake.bsp.bsp29a.Leaf method*), 41
__init__() (*vgio.quake.bsp.bsp29a.Node method*), 40
__init__() (*vgio.quake.dem.Dem method*), 42
__init__() (*vgio.quake.lmp.Lmp method*), 44
__init__() (*vgio.quake.mdl.Frame method*), 50
__init__() (*vgio.quake.mdl.FrameGroup method*), 51
__init__() (*vgio.quake.mdl.Mdl method*), 47
__init__() (*vgio.quake.mdl.Skin method*), 48
__init__() (*vgio.quake.mdl.SkinGroup method*), 48
__init__() (*vgio.quake.mdl.StVertex method*), 49
__init__() (*vgio.quake.mdl.TriVertex method*), 50
__init__() (*vgio.quake.mdl.Triangle method*), 49
__init__() (*vgio.quake.pak.PakFile method*), 52
__init__() (*vgio.quake.pak.PakInfo method*), 54
__init__() (*vgio.quake.protocol.Bad method*), 54
__init__() (*vgio.quake.protocol.CdTrack method*), 65
__init__() (*vgio.quake.protocol.CenterPrint method*), 63
__init__() (*vgio.quake.protocol.ClientData method*), 60
__init__() (*vgio.quake.protocol.CutScene method*), 66
__init__() (*vgio.quake.protocol.Damage method*), 61
__init__() (*vgio.quake.protocol.Disconnect method*), 55
__init__() (*vgio.quake.protocol.Finale method*), 65
__init__() (*vgio.quake.protocol.FoundSecret method*), 64
__init__() (*vgio.quake.protocol.Intermission method*), 65
__init__() (*vgio.quake.protocol.KilledMonster method*), 64
__init__() (*vgio.quake.protocol.LightStyle method*), 58
__init__() (*vgio.quake.protocol.MessageBlock method*), 67
__init__() (*vgio.quake.protocol.Nop method*), 54
__init__() (*vgio.quake.protocol.Particle method*), 61
__init__() (*vgio.quake.protocol.Print method*), 56
__init__() (*vgio.quake.protocol.SellScreen method*), 65
__init__() (*vgio.quake.protocol.ServerInfo method*),

58
__init__() (vgio.quake.protocol.SetAngle method), 57
__init__() (vgio.quake.protocol.SetPause method), 63
__init__() (vgio.quake.protocol.SetView method), 55
__init__() (vgio.quake.protocol.SignOnNum method), 63
__init__() (vgio.quake.protocol.Sound method), 56
__init__() (vgio.quake.protocol.SpawnBaseline method), 62
__init__() (vgio.quake.protocol.SpawnBinary method), 62
__init__() (vgio.quake.protocol.SpawnStatic method), 62
__init__() (vgio.quake.protocol.SpawnStaticSound method), 64
__init__() (vgio.quake.protocol.StopSound method), 60
__init__() (vgio.quake.protocol.StuffText method), 57
__init__() (vgio.quake.protocol.TempEntity method), 63
__init__() (vgio.quake.protocol.Time method), 56
__init__() (vgio.quake.protocol.UpdateColors method), 60
__init__() (vgio.quake.protocol.UpdateEntity method), 66
__init__() (vgio.quake.protocol.UpdateFrags method), 59
__init__() (vgio.quake.protocol.UpdateName method), 58
__init__() (vgio.quake.protocol.UpdateStat method), 55
__init__() (vgio.quake.protocol.Version method), 55
__init__() (vgio.quake.spr.Spr method), 68
__init__() (vgio.quake.spr.SpriteFrame method), 69
__init__() (vgio.quake.spr.SpriteGroup method), 69
__init__() (vgio.quake.wad.WadFile method), 70
__init__() (vgio.quake.wad.WadInfo method), 72
__init__() (vgio.quake2.bsp.Area method), 80
__init__() (vgio.quake2.bsp.AreaPortal method), 80
__init__() (vgio.quake2.bsp.Brush method), 79
__init__() (vgio.quake2.bsp.BrushSide method), 79
__init__() (vgio.quake2.bsp.Bsp method), 74
__init__() (vgio.quake2.bsp.Edge method), 78
__init__() (vgio.quake2.bsp.Face method), 77
__init__() (vgio.quake2.bsp.Leaf method), 78
__init__() (vgio.quake2.bsp.Lump method), 75
__init__() (vgio.quake2.bsp.Model method), 79
__init__() (vgio.quake2.bsp.Node method), 76
__init__() (vgio.quake2.bsp.Plane method), 75
__init__() (vgio.quake2.bsp.TextureInfo method), 77
__init__() (vgio.quake2.bsp.Vertex method), 75
__init__() (vgio.quake2.md2.Frame method), 84
__init__() (vgio.quake2.md2.GlCommand method), 84
__init__() (vgio.quake2.md2.GlVertex method), 84
__init__() (vgio.quake2.md2.Md2 method), 81
__init__() (vgio.quake2.md2.Skin method), 82
__init__() (vgio.quake2.md2.StVertex method), 83
__init__() (vgio.quake2.md2.TriVertex method), 82
__init__() (vgio.quake2.md2.Triangle method), 83
__init__() (vgio.quake2.pak.PakFile method), 85
__init__() (vgio.quake2.pak.PakInfo method), 87
__init__() (vgio.quake2.protocol.Bad method), 87
__init__() (vgio.quake2.protocol.CenterPrint method), 91
__init__() (vgio.quake2.protocol.ConfigString method), 91
__init__() (vgio.quake2.protocol.DeltaPacketEntities method), 92
__init__() (vgio.quake2.protocol.Disconnect method), 89
__init__() (vgio.quake2.protocol.Download method), 92
__init__() (vgio.quake2.protocol.Frame method), 92
__init__() (vgio.quake2.protocol.Inventory method), 88
__init__() (vgio.quake2.protocol.Layout method), 88
__init__() (vgio.quake2.protocol.MessageBlock method), 93
__init__() (vgio.quake2.protocol.MuzzleFlash method), 87
__init__() (vgio.quake2.protocol.MuzzleFlash2 method), 87
__init__() (vgio.quake2.protocol.Nop method), 89
__init__() (vgio.quake2.protocol.PacketEntities method), 92
__init__() (vgio.quake2.protocol.PlayerInfo method), 92
__init__() (vgio.quake2.protocol.Print method), 90
__init__() (vgio.quake2.protocol.Reconnect method), 89
__init__() (vgio.quake2.protocol.ServerData method), 91
__init__() (vgio.quake2.protocol.Sound method), 90
__init__() (vgio.quake2.protocol.SpawnBaseline method), 91
__init__() (vgio.quake2.protocol.StuffText method), 90
__init__() (vgio.quake2.protocol.TempEntity method), 88
__init__() (vgio.quake2.sp2.Sp2 method), 93
__init__() (vgio.quake2.sp2.SpriteFrame method), 94
__init__() (vgio.quake2.wal.Wal method), 96

## A

active_ammo (vgio.quake.protocol.ClientData attribute), 59
active_weapon (vgio.quake.protocol.ClientData attribute), 60

ambient_level (*vgio.quake.bsp.bsp29.Leaf attribute*), 37
ammo (*vgio.quake.protocol.ClientData attribute*), 60
angle (*vgio.duke3d.map.Map attribute*), 20
angle (*vgio.duke3d.map.Sprite attribute*), 23
angles (*vgio.quake.protocol.SetAngle attribute*), 57
angles (*vgio.quake.protocol.SpawnBaseline attribute*), 62
angles (*vgio.quake.protocol.SpawnStatic attribute*), 62
angles (*vgio.quake.protocol.UpdateEntity attribute*), 66
animation_name (*vgio.quake2.wal.Wal attribute*), 95
anorms (*in module vgio.quake*), 72
anorms (*in module vgio.quake2*), 96
ArchiveFile (*class in vgio._core*), 97
ArchiveInfo (*class in vgio._core*), 99
Area (*class in vgio.quake2.bsp*), 80
area (*vgio.quake2.bsp.Leaf attribute*), 78
area_portals (*vgio.quake2.bsp.Bsp attribute*), 74
AreaPortal (*class in vgio.quake2.bsp*), 80
areas (*vgio.quake2.bsp.Bsp attribute*), 74
areas (*vgio.quake2.protocol.Frame attribute*), 92
armor (*vgio.quake.protocol.ClientData attribute*), 59
armor (*vgio.quake.protocol.Damage attribute*), 61
ArtFile (*class in vgio.duke3d.art*), 14
ArtInfo (*class in vgio.duke3d.art*), 16
attenuation (*vgio.quake.protocol.Sound attribute*), 56
attenuation (*vgio.quake.protocol.SpawnStaticSound attribute*), 64
attenuation (*vgio.quake2.protocol.Sound attribute*), 89
attract_loop (*vgio.quake2.protocol.ServerData attribute*), 90

## B

Bad (*class in vgio.quake.protocol*), 54
Bad (*class in vgio.quake2.protocol*), 87
beam_length (*vgio.quake.spr.Spr attribute*), 68
bit_mask (*vgio.quake.protocol.ClientData attribute*), 59
bit_mask (*vgio.quake.protocol.Sound attribute*), 56
bit_mask (*vgio.quake.protocol.UpdateEntity attribute*), 66
blood (*vgio.quake.protocol.Damage attribute*), 61
bounding_box_max (*vgio.hexen2.bsp.Model attribute*), 27
bounding_box_max (*vgio.quake.bsp.bsp29.Leaf attribute*), 37
bounding_box_max (*vgio.quake.bsp.bsp29.Model attribute*), 37
bounding_box_max (*vgio.quake.bsp.bsp29.Node attribute*), 34
bounding_box_max (*vgio.quake.mdl.Frame attribute*), 50
bounding_box_max (*vgio.quake.mdl.FrameGroup attribute*), 51
bounding_box_max (*vgio.quake2.bsp.Leaf attribute*), 78
bounding_box_max (*vgio.quake2.bsp.Model attribute*), 79
bounding_box_max (*vgio.quake2.bsp.Node attribute*), 76
bounding_box_min (*vgio.hexen2.bsp.Model attribute*), 27
bounding_box_min (*vgio.quake.bsp.bsp29.Leaf attribute*), 36
bounding_box_min (*vgio.quake.bsp.bsp29.Model attribute*), 37
bounding_box_min (*vgio.quake.bsp.bsp29.Node attribute*), 34
bounding_box_min (*vgio.quake.mdl.Frame attribute*), 50
bounding_box_min (*vgio.quake.mdl.FrameGroup attribute*), 51
bounding_box_min (*vgio.quake2.bsp.Leaf attribute*), 78
bounding_box_min (*vgio.quake2.bsp.Model attribute*), 79
bounding_box_min (*vgio.quake2.bsp.Node attribute*), 76
bounding_radius (*vgio.quake.mdl.Mdl attribute*), 46
bounding_radius (*vgio.quake.spr.Spr attribute*), 67
Brush (*class in vgio.quake.map*), 45
Brush (*class in vgio.quake2.bsp*), 79
brush_sides (*vgio.quake2.bsp.Bsp attribute*), 74
brushes (*vgio.quake.map.Entity attribute*), 45
brushes (*vgio.quake2.bsp.Bsp attribute*), 73
BrushSide (*class in vgio.quake2.bsp*), 79
Bsp (*class in vgio.hexen2.bsp*), 25
Bsp (*class in vgio.quake.bsp*), 42
Bsp (*class in vgio.quake.bsp.bsp29*), 31
Bsp (*class in vgio.quake.bsp.bsp29a*), 38
Bsp (*class in vgio.quake2.bsp*), 73

## C

cd_track (*vgio.quake.dem.Dem attribute*), 42
CdTrack (*class in vgio.quake.protocol*), 65
ceiling_heinum (*vgio.duke3d.map.Sector attribute*), 21
ceiling_palette (*vgio.duke3d.map.Sector attribute*), 21
ceiling_picnum (*vgio.duke3d.map.Sector attribute*), 21
ceiling_shade (*vgio.duke3d.map.Sector attribute*), 21
ceiling_stat (*vgio.duke3d.map.Sector attribute*), 21
ceiling_x_panning (*vgio.duke3d.map.Sector attribute*), 21
ceiling_y_panning (*vgio.duke3d.map.Sector attribute*), 21
ceiling_z (*vgio.duke3d.map.Sector attribute*), 21
CenterPrint (*class in vgio.quake.protocol*), 63
CenterPrint (*class in vgio.quake2.protocol*), 91
channel (*vgio.quake.protocol.Sound attribute*), 56
channel (*vgio.quake.protocol.StopSound attribute*), 60
channel (*vgio.quake2.protocol.Sound attribute*), 89

children (*vgio.quake.bsp.bsp29.ClipNode attribute*), 36
children (*vgio.quake.bsp.bsp29.Node attribute*), 34
children (*vgio.quake2.bsp.Node attribute*), 76
ClientData (*class in vgio.quake.protocol*), 59
clip_distance (*vgio.duke3d.map.Sprite attribute*), 23
clip_nodes (*vgio.hexen2.bsp.Bsp attribute*), 26
clip_nodes (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
clip_nodes (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
ClipNode (*class in vgio.quake.bsp.bsp29*), 36
ClipNode (*class in vgio.quake.bsp.bsp29a*), 41
close() (*vgio._core.__init__.ArchiveFile method*), 98
close() (*vgio._core.ReadWriteFile method*), 97
close() (*vgio.devildaggers.hxmesh.HxMesh method*), 8
close() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 10
close() (*vgio.devildaggers.hxshader.HxShader method*), 12
close() (*vgio.devildaggers.hxtexture.HxTexture method*), 13
close() (*vgio.duke3d.art.ArtFile method*), 15
close() (*vgio.duke3d.grp.GrpFile method*), 18
close() (*vgio.duke3d.map.Map method*), 21
close() (*vgio.hexen2.bsp.Bsp method*), 27
close() (*vgio.hrot.pak.PakFile method*), 29
close() (*vgio.quake.bsp.bsp29.Bsp method*), 32
close() (*vgio.quake.bsp.bsp29a.Bsp method*), 40
close() (*vgio.quake.dem.Dem method*), 43
close() (*vgio.quake.lmp.Lmp method*), 44
close() (*vgio.quake.mdl.Mdl method*), 47
close() (*vgio.quake.pak.PakFile method*), 52
close() (*vgio.quake.spr.Spr method*), 68
close() (*vgio.quake.wad.WadFile method*), 70
close() (*vgio.quake2.bsp.Bsp method*), 74
close() (*vgio.quake2.md2.Md2 method*), 82
close() (*vgio.quake2.pak.PakFile method*), 85
close() (*vgio.quake2.sp2.Sp2 method*), 94
close() (*vgio.quake2.wal.Wal method*), 96
cluster (*vgio.quake2.bsp.Leaf attribute*), 78
color (*vgio.quake.protocol.Particle attribute*), 61
color_map (*vgio.quake.protocol.SpawnBaseline attribute*), 62
color_map (*vgio.quake.protocol.SpawnStatic attribute*), 61
color_map (*vgio.quake.protocol.UpdateEntity attribute*), 66
colormap (*vgio.quake.lmp.Lmp attribute*), 44
colors (*vgio.quake.protocol.UpdateColors attribute*), 60
compression (*vgio.quake.wad.WadInfo attribute*), 72
ConfigString (*class in vgio.quake2.protocol*), 91
contents (*vgio.quake.bsp.bsp29.Leaf attribute*), 36
contents (*vgio.quake2.bsp.Leaf attribute*), 78
contents (*vgio.quake2.wal.Wal attribute*), 95
count (*vgio.quake.protocol.Particle attribute*), 61
cstat (*vgio.duke3d.map.Sprite attribute*), 22
cstat (*vgio.duke3d.map.Wall attribute*), 24
CutScene (*class in vgio.quake.protocol*), 66

## D

Damage (*class in vgio.quake.protocol*), 61
date_time (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo attribute*), 11
delta_frame (*vgio.quake2.protocol.Frame attribute*), 92
DeltaPacketEntities (*class in vgio.quake2.protocol*), 92
Dem (*class in vgio.quake.dem*), 42
direction (*vgio.quake.protocol.Particle attribute*), 61
Disconnect (*class in vgio.quake.protocol*), 55
Disconnect (*class in vgio.quake2.protocol*), 89
disk_size (*vgio.quake.wad.WadInfo attribute*), 72
distance (*vgio.quake.bsp.bsp29.Plane attribute*), 33
distance (*vgio.quake2.bsp.Plane attribute*), 75
Dm2 (*class in vgio.quake2.dm2*), 80
Download (*class in vgio.quake2.protocol*), 92
dumps() (*in module vgio.quake.map*), 45

## E

Edge (*class in vgio.quake.bsp.bsp29*), 37
Edge (*class in vgio.quake.bsp.bsp29a*), 41
Edge (*class in vgio.quake2.bsp*), 78
edges (*vgio.hexen2.bsp.Bsp attribute*), 26
edges (*vgio.quake.bsp.bsp29.Bsp attribute*), 32
edges (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
edges (*vgio.quake2.bsp.Bsp attribute*), 73
effects (*vgio.quake.protocol.UpdateEntity attribute*), 66
entities (*vgio.hexen2.bsp.Bsp attribute*), 25
entities (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
entities (*vgio.quake.bsp.bsp29a.Bsp attribute*), 38
entities (*vgio.quake2.bsp.Bsp attribute*), 73
Entity (*class in vgio.quake.map*), 45
entity (*vgio.quake.protocol.SetView attribute*), 55
entity (*vgio.quake.protocol.Sound attribute*), 56
entity (*vgio.quake.protocol.SpawnBaseline attribute*), 62
entity (*vgio.quake.protocol.StopSound attribute*), 60
entity (*vgio.quake.protocol.UpdateEntity attribute*), 66
entity (*vgio.quake2.protocol.MuzzleFlash attribute*), 87
entity (*vgio.quake2.protocol.MuzzleFlash2 attribute*), 87
entity (*vgio.quake2.protocol.Sound attribute*), 89
extra (*vgio.duke3d.map.Sector attribute*), 22
extra (*vgio.duke3d.map.Sprite attribute*), 23
extra (*vgio.duke3d.map.Wall attribute*), 24
extract() (*vgio._core.__init__.ArchiveFile method*), 98
extract() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 10
extract() (*vgio.duke3d.art.ArtFile method*), 15
extract() (*vgio.duke3d.grp.GrpFile method*), 18
extract() (*vgio.hrot.pak.PakFile method*), 29

extract() (*vgio.quake.pak.PakFile method*), 53
extract() (*vgio.quake.wad.WadFile method*), 71
extract() (*vgio.quake2.pak.PakFile method*), 86
extractall() (*vgio._core.__init__.ArchiveFile method*), 99
extractall() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 11
extractall() (*vgio.duke3d.art.ArtFile method*), 16
extractall() (*vgio.duke3d.grp.GrpFile method*), 18
extractall() (*vgio.hrot.pak.PakFile method*), 29
extractall() (*vgio.quake.pak.PakFile method*), 53
extractall() (*vgio.quake.wad.WadFile method*), 71
extractall() (*vgio.quake2.pak.PakFile method*), 86
eye_position (*vgio.quake.mdl.Mdl attribute*), 46

## F

Face (*class in vgio.quake.bsp.bsp29*), 35
Face (*class in vgio.quake.bsp.bsp29a*), 40
Face (*class in vgio.quake2.bsp*), 77
faces (*vgio.hexen2.bsp.Bsp attribute*), 26
faces (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
faces (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
faces (*vgio.quake2.bsp.Bsp attribute*), 73
faces_front (*vgio.quake.mdl.Triangle attribute*), 49
file (*vgio._core.ArchiveFile attribute*), 97
file (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile attribute*), 9
file (*vgio.duke3d.art.ArtFile attribute*), 14
file (*vgio.duke3d.grp.GrpFile attribute*), 17
file_offset (*vgio._core.ArchiveInfo attribute*), 99
file_offset (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo attribute*), 11
file_offset (*vgio.duke3d.art.ArtInfo attribute*), 16
file_offset (*vgio.duke3d.grp.GrpInfo attribute*), 19
file_offset (*vgio.hrot.pak.PakInfo attribute*), 30
file_offset (*vgio.quake.pak.PakInfo attribute*), 54
file_offset (*vgio.quake.wad.WadInfo attribute*), 72
file_size (*vgio._core.ArchiveInfo attribute*), 99
file_size (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo attribute*), 11
file_size (*vgio.duke3d.art.ArtInfo attribute*), 16
file_size (*vgio.duke3d.grp.GrpInfo attribute*), 19
file_size (*vgio.hrot.pak.PakInfo attribute*), 30
file_size (*vgio.quake.pak.PakInfo attribute*), 54
file_size (*vgio.quake.wad.WadInfo attribute*), 72
filename (*vgio._core.ArchiveInfo attribute*), 99
filename (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo attribute*), 11
filename (*vgio.duke3d.grp.GrpInfo attribute*), 19
filename (*vgio.hrot.pak.PakInfo attribute*), 30
filename (*vgio.quake.pak.PakInfo attribute*), 54
filename (*vgio.quake.wad.WadInfo attribute*), 72
Finale (*class in vgio.quake.protocol*), 65
first_edge (*vgio.quake.bsp.bsp29.Face attribute*), 35
first_edge (*vgio.quake2.bsp.Face attribute*), 77
first_face (*vgio.hexen2.bsp.Model attribute*), 27
first_face (*vgio.quake.bsp.bsp29.Model attribute*), 38
first_face (*vgio.quake.bsp.bsp29.Node attribute*), 34
first_face (*vgio.quake2.bsp.Model attribute*), 79
first_face (*vgio.quake2.bsp.Node attribute*), 76
first_leaf_brush (*vgio.quake2.bsp.Leaf attribute*), 78
first_leaf_face (*vgio.quake2.bsp.Leaf attribute*), 78
first_mark_surface (*vgio.quake.bsp.bsp29.Leaf attribute*), 37
flags (*vgio.quake.bsp.bsp29.TextureInfo attribute*), 35
flags (*vgio.quake.mdl.Mdl attribute*), 47
flags (*vgio.quake2.bsp.TextureInfo attribute*), 76
flags (*vgio.quake2.protocol.Sound attribute*), 89
flags (*vgio.quake2.wal.Wal attribute*), 95
flash_number (*vgio.quake2.protocol.MuzzleFlash2 attribute*), 87
floor_heinum (*vgio.duke3d.map.Sector attribute*), 22
floor_palette (*vgio.duke3d.map.Sector attribute*), 22
floor_picnum (*vgio.duke3d.map.Sector attribute*), 21
floor_shade (*vgio.duke3d.map.Sector attribute*), 22
floor_stat (*vgio.duke3d.map.Sector attribute*), 21
floor_x_panning (*vgio.duke3d.map.Sector attribute*), 22
floor_y_panning (*vgio.duke3d.map.Sector attribute*), 22
floor_z (*vgio.duke3d.map.Sector attribute*), 21
FoundSecret (*class in vgio.quake.protocol*), 64
fp (*vgio._core.ReadWriteFile attribute*), 97
fp (*vgio.hexen2.bsp.Bsp attribute*), 26
fp (*vgio.quake.bsp.bsp29.Bsp attribute*), 32
fp (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
fp (*vgio.quake.spr.Spr attribute*), 68
fragment_shader (*vgio.devildaggers.hxshader.HxShader attribute*), 12
frags (*vgio.quake.protocol.UpdateFrags attribute*), 59
Frame (*class in vgio.quake.mdl*), 50
Frame (*class in vgio.quake2.md2*), 84
Frame (*class in vgio.quake2.protocol*), 92
frame (*vgio.quake.protocol.SpawnBaseline attribute*), 62
frame (*vgio.quake.protocol.SpawnStatic attribute*), 61
frame (*vgio.quake.protocol.UpdateEntity attribute*), 66
FrameGroup (*class in vgio.quake.mdl*), 51
frames (*vgio.quake.mdl.FrameGroup attribute*), 51
frames (*vgio.quake.mdl.Mdl attribute*), 47
frames (*vgio.quake.spr.SpriteGroup attribute*), 69
frames (*vgio.quake2.md2.Md2 attribute*), 81
frames (*vgio.quake2.sp2.Sp2 attribute*), 93
from_file() (*vgio._core.ArchiveInfo class method*), 99
from_file() (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo class method*), 11
from_track (*vgio.quake.protocol.CdTrack attribute*), 65

## G


game_directory (*vgio.quake2.protocol.ServerData attribute*), 90
getinfo() (*vgio._core.__init__.ArchiveFile method*), 99
getinfo() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 11
getinfo() (*vgio.duke3d.art.ArtFile method*), 16
getinfo() (*vgio.duke3d.grp.GrpFile method*), 18
getinfo() (*vgio.hrot.pak.PakFile method*), 30
getinfo() (*vgio.quake.pak.PakFile method*), 53
getinfo() (*vgio.quake.wad.WadFile method*), 71
getinfo() (*vgio.quake2.pak.PakFile method*), 86
gl_commands (*vgio.quake2.md2.Md2 attribute*), 81
GlCommand (*class in vgio.quake2.md2*), 84
GlVertex (*class in vgio.quake2.md2*), 84
GrpFile (*class in vgio.duke3d.grp*), 17
GrpInfo (*class in vgio.duke3d.grp*), 19


## H


head_node (*vgio.hexen2.bsp.Model attribute*), 27
head_node (*vgio.quake.bsp.bsp29.Model attribute*), 37
head_node (*vgio.quake2.bsp.Model attribute*), 79
health (*vgio.quake.protocol.ClientData attribute*), 59
height (*vgio.devildaggers.hxtexture.HxTexture attribute*), 13
height (*vgio.quake.bsp.bsp29.Miptexture attribute*), 33
height (*vgio.quake.lmp.Lmp attribute*), 44
height (*vgio.quake.spr.Spr attribute*), 67
height (*vgio.quake.spr.SpriteFrame attribute*), 69
height (*vgio.quake2.sp2.SpriteFrame attribute*), 94
height (*vgio.quake2.wal.Wal attribute*), 95
hitag (*vgio.duke3d.map.Sector attribute*), 22
hitag (*vgio.duke3d.map.Sprite attribute*), 23
hitag (*vgio.duke3d.map.Wall attribute*), 24
HxMesh (*class in vgio.devildaggers.hxmesh*), 7
HxResourceGroupFile (*class in vgio.devildaggers.hxresourcegroup*), 9
HxShader (*class in vgio.devildaggers.hxshader*), 12
HxTexture (*class in vgio.devildaggers.hxtexture*), 13


## I


ideal_pitch (*vgio.quake.protocol.ClientData attribute*), 59
identifier (*vgio.quake.mdl.Mdl attribute*), 46
identity (*vgio.quake.spr.Spr attribute*), 67
identity (*vgio.quake2.bsp.Bsp attribute*), 73
identity (*vgio.quake2.md2.Md2 attribute*), 81
identity (*vgio.quake2.sp2.Sp2 attribute*), 93
in_water (*vgio.quake.protocol.ClientData attribute*), 59
index (*vgio.quake.protocol.UpdateStat attribute*), 55
indices (*vgio.devildaggers.hxmesh.HxMesh attribute*), 7
infolist() (*vgio._core.__init__.ArchiveFile method*), 99
infolist() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 11
infolist() (*vgio.duke3d.art.ArtFile method*), 16
infolist() (*vgio.duke3d.grp.GrpFile method*), 19
infolist() (*vgio.hrot.pak.PakFile method*), 30
infolist() (*vgio.quake.pak.PakFile method*), 53
infolist() (*vgio.quake.wad.WadFile method*), 71
infolist() (*vgio.quake2.pak.PakFile method*), 86
Intermission (*class in vgio.quake.protocol*), 65
intervals (*vgio.quake.mdl.FrameGroup attribute*), 51
intervals (*vgio.quake.mdl.SkinGroup attribute*), 48
intervals (*vgio.quake.spr.SpriteGroup attribute*), 69
Inventory (*class in vgio.quake2.protocol*), 88
inventory (*vgio.quake2.protocol.Inventory attribute*), 88
is_artfile() (*in module vgio.duke3d.art*), 14
is_bspfile() (*in module vgio.hexen2.bsp*), 25
is_bspfile() (*in module vgio.quake.bsp*), 41
is_bspfile() (*in module vgio.quake.bsp.bsp29*), 31
is_bspfile() (*in module vgio.quake.bsp.bsp29a*), 38
is_bspfile() (*in module vgio.quake2.bsp*), 72
is_grpfile() (*in module vgio.duke3d.grp*), 17
is_mapfile() (*in module vgio.duke3d.map*), 19
is_md2file() (*in module vgio.quake2.md2*), 80
is_mdlfile() (*in module vgio.quake.mdl*), 46
is_pakfile() (*in module vgio.hrot.pak*), 28
is_pakfile() (*in module vgio.quake.pak*), 51
is_pakfile() (*in module vgio.quake2.pak*), 84
is_sp2file() (*in module vgio.quake2.sp2*), 93
is_sprfile() (*in module vgio.quake.spr*), 67
is_wadfile() (*in module vgio.quake.wad*), 69
item_bit_mask (*vgio.quake.protocol.ClientData attribute*), 59


## K


KilledMonster (*class in vgio.quake.protocol*), 64


## L


Layout (*class in vgio.quake2.protocol*), 88
Leaf (*class in vgio.quake.bsp.bsp29*), 36
Leaf (*class in vgio.quake.bsp.bsp29a*), 41
Leaf (*class in vgio.quake2.bsp*), 78
leaf_brushes (*vgio.quake2.bsp.Bsp attribute*), 73
leaf_faces (*vgio.quake2.bsp.Bsp attribute*), 73
leafs (*vgio.hexen2.bsp.Bsp attribute*), 26
leafs (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
leafs (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
leafs (*vgio.quake2.bsp.Bsp attribute*), 73
length (*vgio.quake2.bsp.Lump attribute*), 75
level (*vgio.quake2.protocol.Print attribute*), 90
light_normal_index (*vgio.quake.mdl.TriVertex attribute*), 50
light_normal_index (*vgio.quake2.md2.TriVertex attribute*), 82

light_offset (*vgio.quake.bsp.bsp29.Face attribute*), 36
light_offset (*vgio.quake2.bsp.Face attribute*), 77
lighting (*vgio.hexen2.bsp.Bsp attribute*), 26
lighting (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
lighting (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
lighting (*vgio.quake2.bsp.Bsp attribute*), 73
LightStyle (*class in vgio.quake.protocol*), 58
Lmp (*class in vgio.quake.lmp*), 43
loads() (*in module vgio.quake.map*), 45
lotag (*vgio.duke3d.map.Sector attribute*), 22
lotag (*vgio.duke3d.map.Sprite attribute*), 23
lotag (*vgio.duke3d.map.Wall attribute*), 24
Lump (*class in vgio.quake2.bsp*), 75

## M

Map (*class in vgio.duke3d.map*), 20
map_name (*vgio.quake.protocol.ServerInfo attribute*), 57
map_name (*vgio.quake2.protocol.ServerData attribute*), 91
mark_surfaces (*vgio.hexen2.bsp.Bsp attribute*), 26
mark_surfaces (*vgio.quake.bsp.bsp29.Bsp attribute*), 32
mark_surfaces (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
max_clients (*vgio.quake.protocol.ServerInfo attribute*), 57
Md2 (*class in vgio.quake2.md2*), 81
Mdl (*class in vgio.quake.mdl*), 46
message_blocks (*vgio.quake.dem.Dem attribute*), 42
message_blocks (*vgio.quake2.dm2.Dm2 attribute*), 80
MessageBlock (*class in vgio.quake.protocol*), 67
MessageBlock (*class in vgio.quake2.protocol*), 93
messages (*vgio.quake.protocol.MessageBlock attribute*), 67
messages (*vgio.quake2.protocol.MessageBlock attribute*), 93
mip_level_count (*vgio.devildaggers.hxtexture.HxTexture attribute*), 13
Miptexture (*class in vgio.quake.bsp.bsp29*), 33
miptexture_number (*vgio.quake.bsp.bsp29.TextureInfo attribute*), 35
miptextures (*vgio.hexen2.bsp.Bsp attribute*), 25
miptextures (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
miptextures (*vgio.quake.bsp.bsp29a.Bsp attribute*), 38
mode (*vgio._core.ArchiveFile attribute*), 97
mode (*vgio._core.ReadWriteFile attribute*), 97
mode (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile attribute*), 9
mode (*vgio.duke3d.art.ArtFile attribute*), 14
mode (*vgio.duke3d.grp.GrpFile attribute*), 17
mode (*vgio.hexen2.bsp.Bsp attribute*), 26
mode (*vgio.quake.bsp.bsp29.Bsp attribute*), 32
mode (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
mode (*vgio.quake.spr.Spr attribute*), 68
Model (*class in vgio.hexen2.bsp*), 27
Model (*class in vgio.quake.bsp.bsp29*), 37
Model (*class in vgio.quake2.bsp*), 79
model_index (*vgio.quake.protocol.SpawnBaseline attribute*), 62
model_index (*vgio.quake.protocol.SpawnStatic attribute*), 61
model_index (*vgio.quake.protocol.UpdateEntity attribute*), 66
models (*vgio.hexen2.bsp.Bsp attribute*), 26
models (*vgio.quake.bsp.bsp29.Bsp attribute*), 32
models (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
models (*vgio.quake.protocol.ServerInfo attribute*), 57
models (*vgio.quake2.bsp.Bsp attribute*), 73
module
    vgio._core, 96
    vgio._core.__init__, 96
    vgio.devildaggers.hxmesh, 7
    vgio.devildaggers.hxresourcegroup, 9
    vgio.devildaggers.hxshader, 12
    vgio.devildaggers.hxtexture, 13
    vgio.duke3d, 14
    vgio.duke3d.art, 14
    vgio.duke3d.grp, 16
    vgio.duke3d.map, 19
    vgio.hexen2, 25
    vgio.hexen2.bsp, 25
    vgio.hrot, 27
    vgio.hrot.pak, 28
    vgio.quake, 30
    vgio.quake.bsp, 30
    vgio.quake.bsp.bsp29, 30
    vgio.quake.bsp.bsp29a, 38
    vgio.quake.lmp, 43
    vgio.quake.mdl, 45
    vgio.quake.pak, 51
    vgio.quake.protocol, 54
    vgio.quake.spr, 67
    vgio.quake.wad, 69
    vgio.quake2, 72
    vgio.quake2.bsp, 72
    vgio.quake2.dm2, 80
    vgio.quake2.md2, 80
    vgio.quake2.pak, 84
    vgio.quake2.protocol, 87
    vgio.quake2.sp2, 93
    vgio.quake2.wal, 94
multi (*vgio.quake.protocol.ServerInfo attribute*), 57
MuzzleFlash (*class in vgio.quake2.protocol*), 87
MuzzleFlash2 (*class in vgio.quake2.protocol*), 87

## N

name (*vgio.devildaggers.hxshader.HxShader attribute*), 12

name (*vgio.quake.bsp.bsp29.Miptexture attribute*), 33
name (*vgio.quake.mdl.Frame attribute*), 50
name (*vgio.quake.protocol.UpdateName attribute*), 58
name (*vgio.quake2.md2.Frame attribute*), 84
name (*vgio.quake2.sp2.SpriteFrame attribute*), 94
name (*vgio.quake2.wal.Wal attribute*), 95
namelist() (*vgio._core.__init__.ArchiveFile method*), 99
namelist() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 11
namelist() (*vgio.duke3d.art.ArtFile method*), 16
namelist() (*vgio.duke3d.grp.GrpFile method*), 19
namelist() (*vgio.hrot.pak.PakFile method*), 30
namelist() (*vgio.quake.pak.PakFile method*), 53
namelist() (*vgio.quake.wad.WadFile method*), 71
namelist() (*vgio.quake2.pak.PakFile method*), 86
next_sector (*vgio.duke3d.map.Wall attribute*), 24
next_texture_info (*vgio.quake2.bsp.TextureInfo attribute*), 77
next_wall (*vgio.duke3d.map.Wall attribute*), 24
Node (*class in vgio.quake.bsp.bsp29*), 34
Node (*class in vgio.quake.bsp.bsp29a*), 40
Node (*class in vgio.quake2.bsp*), 76
nodes (*vgio.hexen2.bsp.Bsp attribute*), 25
nodes (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
nodes (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
nodes (*vgio.quake2.bsp.Bsp attribute*), 73
Nop (*class in vgio.quake.protocol*), 54
Nop (*class in vgio.quake2.protocol*), 89
normal (*vgio.devildaggers.hxmesh.Vertex attribute*), 8
normal (*vgio.quake.bsp.bsp29.Plane attribute*), 33
normal (*vgio.quake2.bsp.Plane attribute*), 75
number_of_edges (*vgio.quake.bsp.bsp29.Face attribute*), 36
number_of_edges (*vgio.quake2.bsp.Face attribute*), 77
number_of_faces (*vgio.hexen2.bsp.Model attribute*), 27
number_of_faces (*vgio.quake.bsp.bsp29.Model attribute*), 38
number_of_faces (*vgio.quake.bsp.bsp29.Node attribute*), 35
number_of_faces (*vgio.quake2.bsp.Model attribute*), 79
number_of_faces (*vgio.quake2.bsp.Node attribute*), 76
number_of_frames (*vgio.quake.mdl.FrameGroup attribute*), 51
number_of_frames (*vgio.quake.mdl.Mdl attribute*), 47
number_of_frames (*vgio.quake.spr.Spr attribute*), 68
number_of_frames (*vgio.quake.spr.SpriteGroup attribute*), 69
number_of_frames (*vgio.quake2.sp2.Sp2 attribute*), 93
number_of_leaf_brushes (*vgio.quake2.bsp.Leaf attribute*), 78
number_of_leaf_faces (*vgio.quake2.bsp.Leaf attribute*), 78
number_of_marked_surfaces (*vgio.quake.bsp.bsp29.Leaf attribute*), 37
number_of_skins (*vgio.quake.mdl.Mdl attribute*), 46
number_of_skins (*vgio.quake.mdl.SkinGroup attribute*), 48
number_of_triangles (*vgio.quake.mdl.Mdl attribute*), 46
number_of_vertexes (*vgio.quake.mdl.Mdl attribute*), 46

## O

offset (*vgio.quake.map.Plane attribute*), 45
offset (*vgio.quake2.bsp.Lump attribute*), 75
offset (*vgio.quake2.protocol.Sound attribute*), 89
offsets (*vgio.quake.bsp.bsp29.Miptexture attribute*), 33
offsets (*vgio.quake2.wal.Wal attribute*), 95
on_ground (*vgio.quake.protocol.ClientData attribute*), 59
on_seam (*vgio.quake.mdl.StVertex attribute*), 49
open() (*vgio._core.__init__.ArchiveFile method*), 97
open() (*vgio._core.ReadWriteFile class method*), 97
open() (*vgio.devildaggers.hxmesh.HxMesh class method*), 7
open() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 9
open() (*vgio.devildaggers.hxshader.HxShader class method*), 12
open() (*vgio.devildaggers.hxtexture.HxTexture class method*), 13
open() (*vgio.duke3d.art.ArtFile method*), 14
open() (*vgio.duke3d.grp.GrpFile method*), 17
open() (*vgio.duke3d.map.Map class method*), 20
open() (*vgio.hexen2.bsp.Bsp class method*), 26
open() (*vgio.hrot.pak.PakFile method*), 28
open() (*vgio.quake.bsp.Bsp static method*), 42
open() (*vgio.quake.bsp.bsp29.Bsp class method*), 32
open() (*vgio.quake.bsp.bsp29a.Bsp class method*), 39
open() (*vgio.quake.dem.Dem class method*), 42
open() (*vgio.quake.lmp.Lmp class method*), 44
open() (*vgio.quake.mdl.Mdl class method*), 47
open() (*vgio.quake.pak.PakFile method*), 52
open() (*vgio.quake.spr.Spr class method*), 68
open() (*vgio.quake.wad.WadFile method*), 70
open() (*vgio.quake2.bsp.Bsp class method*), 74
open() (*vgio.quake2.md2.Md2 class method*), 81
open() (*vgio.quake2.pak.PakFile method*), 85
open() (*vgio.quake2.sp2.Sp2 class method*), 94
open() (*vgio.quake2.wal.Wal class method*), 96
origin (*vgio.hexen2.bsp.Model attribute*), 27
origin (*vgio.quake.bsp.bsp29.Model attribute*), 37
origin (*vgio.quake.mdl.Mdl attribute*), 46
origin (*vgio.quake.protocol.Damage attribute*), 61
origin (*vgio.quake.protocol.Particle attribute*), 61
origin (*vgio.quake.protocol.Sound attribute*), 56

origin (*vgio.quake.protocol.SpawnBaseline attribute*), 62
origin (*vgio.quake.protocol.SpawnStatic attribute*), 62
origin (*vgio.quake.protocol.SpawnStaticSound attribute*), 64
origin (*vgio.quake.protocol.UpdateEntity attribute*), 66
origin (*vgio.quake.spr.SpriteFrame attribute*), 69
origin (*vgio.quake2.bsp.Model attribute*), 79
origin (*vgio.quake2.sp2.SpriteFrame attribute*), 94
over_picnum (*vgio.duke3d.map.Wall attribute*), 24
owner (*vgio.duke3d.map.Sprite attribute*), 23

## P

PacketEntities (*class in vgio.quake2.protocol*), 92
PakFile (*class in vgio.hrot.pak*), 28
PakFile (*class in vgio.quake.pak*), 52
PakFile (*class in vgio.quake2.pak*), 85
PakInfo (*class in vgio.hrot.pak*), 30
PakInfo (*class in vgio.quake.pak*), 54
PakInfo (*class in vgio.quake2.pak*), 87
palette (*in module vgio.duke3d*), 24
palette (*in module vgio.quake*), 72
palette (*vgio.duke3d.map.Sprite attribute*), 22
palette (*vgio.duke3d.map.Wall attribute*), 24
palette (*vgio.quake.lmp.Lmp attribute*), 44
Particle (*class in vgio.quake.protocol*), 61
paused (*vgio.quake.protocol.SetPause attribute*), 63
picanim (*vgio.duke3d.art.ArtInfo attribute*), 16
picnum (*vgio.duke3d.map.Wall attribute*), 24
pixels (*vgio.devildaggers.hxtexture.HxTexture attribute*), 13
pixels (*vgio.quake.bsp.bsp29.Miptexture attribute*), 33
pixels (*vgio.quake.lmp.Lmp attribute*), 44
pixels (*vgio.quake.mdl.Skin attribute*), 48
pixels (*vgio.quake.mdl.SkinGroup attribute*), 48
pixels (*vgio.quake.spr.SpriteFrame attribute*), 69
pixels (*vgio.quake2.wal.Wal attribute*), 95
Plane (*class in vgio.quake.bsp.bsp29*), 33
Plane (*class in vgio.quake.map*), 45
Plane (*class in vgio.quake2.bsp*), 75
plane_number (*vgio.quake.bsp.bsp29.ClipNode attribute*), 36
plane_number (*vgio.quake.bsp.bsp29.Face attribute*), 35
plane_number (*vgio.quake.bsp.bsp29.Node attribute*), 34
plane_number (*vgio.quake2.bsp.Face attribute*), 77
plane_number (*vgio.quake2.bsp.Node attribute*), 76
planes (*vgio.hexen2.bsp.Bsp attribute*), 25
planes (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
planes (*vgio.quake.bsp.bsp29a.Bsp attribute*), 38
planes (*vgio.quake.map.Brush attribute*), 45
planes (*vgio.quake2.bsp.Bsp attribute*), 73
player (*vgio.quake.protocol.UpdateColors attribute*), 60
player (*vgio.quake.protocol.UpdateFrags attribute*), 59
player (*vgio.quake.protocol.UpdateName attribute*), 58
player_number (*vgio.quake2.protocol.ServerData attribute*), 90
PlayerInfo (*class in vgio.quake2.protocol*), 92
point2 (*vgio.duke3d.map.Wall attribute*), 24
points (*vgio.quake.map.Plane attribute*), 45
pop (*vgio.quake2.bsp.Bsp attribute*), 74
position (*vgio.devildaggers.hxmesh.Vertex attribute*), 8
position (*vgio.quake2.protocol.Sound attribute*), 89
position_x (*vgio.duke3d.map.Map attribute*), 20
position_y (*vgio.duke3d.map.Map attribute*), 20
position_z (*vgio.duke3d.map.Map attribute*), 20
Print (*class in vgio.quake.protocol*), 56
Print (*class in vgio.quake2.protocol*), 90
protocol_version (*vgio.quake.protocol.ServerInfo attribute*), 57
protocol_version (*vgio.quake.protocol.Version attribute*), 55
protocol_version (*vgio.quake2.protocol.ServerData attribute*), 90
punch_angle (*vgio.quake.protocol.ClientData attribute*), 59

## R

read() (*vgio._core.__init__.ArchiveFile method*), 98
read() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 10
read() (*vgio.duke3d.art.ArtFile method*), 15
read() (*vgio.duke3d.grp.GrpFile method*), 18
read() (*vgio.duke3d.map.Sector class method*), 22
read() (*vgio.duke3d.map.Sprite class method*), 23
read() (*vgio.duke3d.map.Wall class method*), 24
read() (*vgio.hexen2.bsp.Model class method*), 27
read() (*vgio.hrot.pak.PakFile method*), 29
read() (*vgio.quake.bsp.bsp29.ClipNode class method*), 36
read() (*vgio.quake.bsp.bsp29.Edge class method*), 37
read() (*vgio.quake.bsp.bsp29.Face class method*), 36
read() (*vgio.quake.bsp.bsp29.Leaf class method*), 37
read() (*vgio.quake.bsp.bsp29.Miptexture class method*), 34
read() (*vgio.quake.bsp.bsp29.Model class method*), 38
read() (*vgio.quake.bsp.bsp29.Node class method*), 35
read() (*vgio.quake.bsp.bsp29.Plane class method*), 33
read() (*vgio.quake.bsp.bsp29.TextureInfo class method*), 35
read() (*vgio.quake.bsp.bsp29.Vertex class method*), 34
read() (*vgio.quake.bsp.bsp29a.ClipNode class method*), 41
read() (*vgio.quake.bsp.bsp29a.Edge class method*), 41
read() (*vgio.quake.bsp.bsp29a.Face class method*), 40
read() (*vgio.quake.bsp.bsp29a.Leaf class method*), 41
read() (*vgio.quake.bsp.bsp29a.Node class method*), 40
read() (*vgio.quake.mdl.Frame static method*), 51

read() (*vgio.quake.mdl.FrameGroup static method*), 51
read() (*vgio.quake.mdl.Skin static method*), 48
read() (*vgio.quake.mdl.SkinGroup static method*), 48
read() (*vgio.quake.mdl.StVertex class method*), 49
read() (*vgio.quake.mdl.Triangle class method*), 49
read() (*vgio.quake.mdl.TriVertex class method*), 50
read() (*vgio.quake.pak.PakFile method*), 52
read() (*vgio.quake.protocol.Bad static method*), 54
read() (*vgio.quake.protocol.CdTrack static method*), 65
read() (*vgio.quake.protocol.CenterPrint static method*), 63
read() (*vgio.quake.protocol.ClientData static method*), 60
read() (*vgio.quake.protocol.CutScene static method*), 66
read() (*vgio.quake.protocol.Damage static method*), 61
read() (*vgio.quake.protocol.Disconnect static method*), 55
read() (*vgio.quake.protocol.Finale static method*), 65
read() (*vgio.quake.protocol.FoundSecret static method*), 64
read() (*vgio.quake.protocol.Intermission static method*), 65
read() (*vgio.quake.protocol.KilledMonster static method*), 64
read() (*vgio.quake.protocol.LightStyle static method*), 58
read() (*vgio.quake.protocol.MessageBlock static method*), 67
read() (*vgio.quake.protocol.Nop static method*), 54
read() (*vgio.quake.protocol.Particle static method*), 61
read() (*vgio.quake.protocol.Print static method*), 56
read() (*vgio.quake.protocol.SellScreen static method*), 65
read() (*vgio.quake.protocol.ServerInfo static method*), 58
read() (*vgio.quake.protocol.SetAngle static method*), 57
read() (*vgio.quake.protocol.SetPause static method*), 63
read() (*vgio.quake.protocol.SetView static method*), 55
read() (*vgio.quake.protocol.SignOnNum static method*), 63
read() (*vgio.quake.protocol.Sound static method*), 56
read() (*vgio.quake.protocol.SpawnBaseline static method*), 62
read() (*vgio.quake.protocol.SpawnBinary static method*), 62
read() (*vgio.quake.protocol.SpawnStatic static method*), 62
read() (*vgio.quake.protocol.SpawnStaticSound static method*), 64
read() (*vgio.quake.protocol.StopSound static method*), 60
read() (*vgio.quake.protocol.StuffText static method*), 57
read() (*vgio.quake.protocol.TempEntity static method*), 63
read() (*vgio.quake.protocol.Time static method*), 56
read() (*vgio.quake.protocol.UpdateColors static method*), 60
read() (*vgio.quake.protocol.UpdateEntity static method*), 66
read() (*vgio.quake.protocol.UpdateFrags static method*), 59
read() (*vgio.quake.protocol.UpdateName static method*), 58
read() (*vgio.quake.protocol.UpdateStat static method*), 55
read() (*vgio.quake.protocol.Version static method*), 55
read() (*vgio.quake.spr.SpriteFrame static method*), 69
read() (*vgio.quake.spr.SpriteGroup static method*), 69
read() (*vgio.quake.wad.WadFile method*), 70
read() (*vgio.quake2.bsp.Area class method*), 80
read() (*vgio.quake2.bsp.AreaPortal class method*), 80
read() (*vgio.quake2.bsp.Brush class method*), 79
read() (*vgio.quake2.bsp.BrushSide class method*), 79
read() (*vgio.quake2.bsp.Edge class method*), 78
read() (*vgio.quake2.bsp.Face class method*), 77
read() (*vgio.quake2.bsp.Leaf class method*), 78
read() (*vgio.quake2.bsp.Lump class method*), 75
read() (*vgio.quake2.bsp.Model class method*), 79
read() (*vgio.quake2.bsp.Node class method*), 76
read() (*vgio.quake2.bsp.Plane class method*), 75
read() (*vgio.quake2.bsp.TextureInfo class method*), 77
read() (*vgio.quake2.bsp.Vertex class method*), 75
read() (*vgio.quake2.md2.Frame class method*), 84
read() (*vgio.quake2.md2.GlCommand class method*), 84
read() (*vgio.quake2.md2.GlVertex class method*), 84
read() (*vgio.quake2.md2.Skin class method*), 82
read() (*vgio.quake2.md2.StVertex class method*), 83
read() (*vgio.quake2.md2.Triangle class method*), 83
read() (*vgio.quake2.md2.TriVertex class method*), 83
read() (*vgio.quake2.pak.PakFile method*), 85
read() (*vgio.quake2.protocol.Bad class method*), 87
read() (*vgio.quake2.protocol.CenterPrint class method*), 91
read() (*vgio.quake2.protocol.ConfigString class method*), 91
read() (*vgio.quake2.protocol.DeltaPacketEntities class method*), 92
read() (*vgio.quake2.protocol.Disconnect class method*), 89
read() (*vgio.quake2.protocol.Download class method*), 92
read() (*vgio.quake2.protocol.Frame class method*), 92
read() (*vgio.quake2.protocol.Inventory class method*), 88
read() (*vgio.quake2.protocol.Layout class method*), 88
read() (*vgio.quake2.protocol.MessageBlock static method*), 93

read() (*vgio.quake2.protocol.MuzzleFlash class method*), 87
read() (*vgio.quake2.protocol.MuzzleFlash2 class method*), 88
read() (*vgio.quake2.protocol.Nop class method*), 89
read() (*vgio.quake2.protocol.PacketEntities class method*), 92
read() (*vgio.quake2.protocol.PlayerInfo class method*), 92
read() (*vgio.quake2.protocol.Print class method*), 90
read() (*vgio.quake2.protocol.Reconnect class method*), 89
read() (*vgio.quake2.protocol.ServerData class method*), 91
read() (*vgio.quake2.protocol.Sound class method*), 90
read() (*vgio.quake2.protocol.SpawnBaseline class method*), 91
read() (*vgio.quake2.protocol.StuffText class method*), 90
read() (*vgio.quake2.protocol.TempEntity class method*), 88
ReadWriteFile (*class in vgio._core*), 97
Reconnect (*class in vgio.quake2.protocol*), 89
ResourceGroupInfo (*class in vgio.devildaggers.hxresourcegroup*), 11
rotation (*vgio.quake.map.Plane attribute*), 45

## S

s (*vgio.quake.bsp.bsp29.TextureInfo attribute*), 35
s (*vgio.quake.mdl.StVertex attribute*), 49
s (*vgio.quake2.bsp.TextureInfo attribute*), 76
s (*vgio.quake2.md2.StVertex attribute*), 83
s_offset (*vgio.quake.bsp.bsp29.TextureInfo attribute*), 35
s_offset (*vgio.quake2.bsp.TextureInfo attribute*), 76
save() (*vgio._core.ReadWriteFile method*), 97
save() (*vgio.devildaggers.hxmesh.HxMesh method*), 8
save() (*vgio.devildaggers.hxshader.HxShader method*), 12
save() (*vgio.devildaggers.hxtexture.HxTexture method*), 14
save() (*vgio.duke3d.map.Map method*), 21
save() (*vgio.hexen2.bsp.Bsp method*), 27
save() (*vgio.quake.bsp.bsp29.Bsp method*), 32
save() (*vgio.quake.bsp.bsp29a.Bsp method*), 40
save() (*vgio.quake.dem.Dem method*), 43
save() (*vgio.quake.lmp.Lmp method*), 44
save() (*vgio.quake.mdl.Mdl method*), 47
save() (*vgio.quake.spr.Spr method*), 68
save() (*vgio.quake2.bsp.Bsp method*), 74
save() (*vgio.quake2.md2.Md2 method*), 82
save() (*vgio.quake2.sp2.Sp2 method*), 94
save() (*vgio.quake2.wal.Wal method*), 96
scale (*vgio.quake.map.Plane attribute*), 45
scale (*vgio.quake.mdl.Mdl attribute*), 46
scale (*vgio.quake2.md2.Frame attribute*), 84
Sector (*class in vgio.duke3d.map*), 21
sector_number (*vgio.duke3d.map.Sprite attribute*), 23
sectors (*vgio.duke3d.map.Map attribute*), 20
SellScreen (*class in vgio.quake.protocol*), 65
server_count (*vgio.quake2.protocol.ServerData attribute*), 90
server_frame (*vgio.quake2.protocol.Frame attribute*), 92
ServerData (*class in vgio.quake2.protocol*), 90
ServerInfo (*class in vgio.quake.protocol*), 57
SetAngle (*class in vgio.quake.protocol*), 57
SetPause (*class in vgio.quake.protocol*), 63
SetView (*class in vgio.quake.protocol*), 55
shade (*vgio.duke3d.map.Sprite attribute*), 22
shade (*vgio.duke3d.map.Wall attribute*), 24
side (*vgio.quake.bsp.bsp29.Face attribute*), 35
side (*vgio.quake2.bsp.Face attribute*), 77
sign_on (*vgio.quake.protocol.SignOnNum attribute*), 63
SignOnNum (*class in vgio.quake.protocol*), 63
size (*vgio.quake.mdl.Mdl attribute*), 47
Skin (*class in vgio.quake.mdl*), 48
Skin (*class in vgio.quake2.md2*), 82
skin (*vgio.quake.protocol.SpawnBaseline attribute*), 62
skin (*vgio.quake.protocol.SpawnStatic attribute*), 61
skin (*vgio.quake.protocol.UpdateEntity attribute*), 66
skin_height (*vgio.quake.mdl.Mdl attribute*), 46
skin_height (*vgio.quake2.md2.Md2 attribute*), 81
skin_width (*vgio.quake.mdl.Mdl attribute*), 46
skin_width (*vgio.quake2.md2.Md2 attribute*), 81
SkinGroup (*class in vgio.quake.mdl*), 48
skins (*vgio.quake.mdl.Mdl attribute*), 47
skins (*vgio.quake2.md2.Md2 attribute*), 81
Sound (*class in vgio.quake.protocol*), 56
Sound (*class in vgio.quake2.protocol*), 89
sound_number (*vgio.quake.protocol.Sound attribute*), 56
sound_number (*vgio.quake.protocol.SpawnStaticSound attribute*), 64
sound_number (*vgio.quake2.protocol.Sound attribute*), 89
sounds (*vgio.quake.protocol.ServerInfo attribute*), 57
Sp2 (*class in vgio.quake2.sp2*), 93
SpawnBaseline (*class in vgio.quake.protocol*), 62
SpawnBaseline (*class in vgio.quake2.protocol*), 91
SpawnBinary (*class in vgio.quake.protocol*), 62
SpawnStatic (*class in vgio.quake.protocol*), 61
SpawnStaticSound (*class in vgio.quake.protocol*), 64
Spr (*class in vgio.quake.spr*), 67
Sprite (*class in vgio.duke3d.map*), 22
SpriteFrame (*class in vgio.quake.spr*), 69
SpriteFrame (*class in vgio.quake2.sp2*), 94
SpriteGroup (*class in vgio.quake.spr*), 69
sprites (*vgio.duke3d.map.Map attribute*), 20
st_vertexes (*vgio.quake.mdl.Mdl attribute*), 47

st_vertexes (*vgio.quake2.md2.Md2 attribute*), 81
start_sector (*vgio.duke3d.map.Map attribute*), 20
status_number (*vgio.duke3d.map.Sprite attribute*), 23
StopSound (*class in vgio.quake.protocol*), 60
string (*vgio.quake.protocol.LightStyle attribute*), 58
StuffText (*class in vgio.quake.protocol*), 57
StuffText (*class in vgio.quake2.protocol*), 90
StVertex (*class in vgio.quake.mdl*), 49
StVertex (*class in vgio.quake2.md2*), 83
style (*vgio.quake.protocol.LightStyle attribute*), 58
styles (*vgio.quake.bsp.bsp29.Face attribute*), 36
styles (*vgio.quake2.bsp.Face attribute*), 77
surf_edges (*vgio.hexen2.bsp.Bsp attribute*), 26
surf_edges (*vgio.quake.bsp.bsp29.Bsp attribute*), 32
surf_edges (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
surf_edges (*vgio.quake2.bsp.Bsp attribute*), 73
sync_type (*vgio.quake.spr.Spr attribute*), 68
synctype (*vgio.quake.mdl.Mdl attribute*), 47

## T

t (*vgio.quake.bsp.bsp29.TextureInfo attribute*), 35
t (*vgio.quake.mdl.StVertex attribute*), 49
t (*vgio.quake2.bsp.TextureInfo attribute*), 76
t (*vgio.quake2.md2.StVertex attribute*), 83
t_offset (*vgio.quake.bsp.bsp29.TextureInfo attribute*), 35
t_offset (*vgio.quake2.bsp.TextureInfo attribute*), 76
TempEntity (*class in vgio.quake.protocol*), 63
TempEntity (*class in vgio.quake2.protocol*), 88
text (*vgio.quake.protocol.CenterPrint attribute*), 63
text (*vgio.quake.protocol.CutScene attribute*), 66
text (*vgio.quake.protocol.Finale attribute*), 65
text (*vgio.quake.protocol.Print attribute*), 56
text (*vgio.quake.protocol.StuffText attribute*), 57
text (*vgio.quake2.protocol.Layout attribute*), 88
text (*vgio.quake2.protocol.Print attribute*), 90
text (*vgio.quake2.protocol.StuffText attribute*), 90
texture_format (*vgio.devildaggers.hxtexture.HxTexture attribute*), 13
texture_info (*vgio.quake.bsp.bsp29.Face attribute*), 36
texture_info (*vgio.quake2.bsp.Face attribute*), 77
texture_infos (*vgio.hexen2.bsp.Bsp attribute*), 25
texture_infos (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
texture_infos (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
texture_infos (*vgio.quake2.bsp.Bsp attribute*), 73
texture_name (*vgio.quake.map.Plane attribute*), 45
texture_name (*vgio.quake2.bsp.TextureInfo attribute*), 77
TextureInfo (*class in vgio.quake.bsp.bsp29*), 35
TextureInfo (*class in vgio.quake2.bsp*), 76
tile_dimensions (*vgio.duke3d.art.ArtInfo attribute*), 16
tile_index (*vgio.duke3d.art.ArtInfo attribute*), 16
Time (*class in vgio.quake.protocol*), 56
time (*vgio.quake.protocol.Time attribute*), 56
to_track (*vgio.quake.protocol.CdTrack attribute*), 65
translate (*vgio.quake2.md2.Frame attribute*), 84
Triangle (*class in vgio.quake.mdl*), 49
Triangle (*class in vgio.quake2.md2*), 83
triangles (*vgio.quake.mdl.Mdl attribute*), 47
triangles (*vgio.quake2.md2.Md2 attribute*), 81
TriVertex (*class in vgio.quake.mdl*), 50
TriVertex (*class in vgio.quake2.md2*), 82
type (*vgio.devildaggers.hxresourcegroup.ResourceGroupInfo attribute*), 11
type (*vgio.quake.bsp.bsp29.Plane attribute*), 33
type (*vgio.quake.mdl.Frame attribute*), 50
type (*vgio.quake.mdl.FrameGroup attribute*), 51
type (*vgio.quake.mdl.Skin attribute*), 48
type (*vgio.quake.mdl.SkinGroup attribute*), 48
type (*vgio.quake.protocol.TempEntity attribute*), 63
type (*vgio.quake.spr.Spr attribute*), 67
type (*vgio.quake.wad.WadInfo attribute*), 72
type (*vgio.quake2.bsp.Plane attribute*), 75
type (*vgio.quake2.protocol.TempEntity attribute*), 88

## U

UpdateColors (*class in vgio.quake.protocol*), 60
UpdateEntity (*class in vgio.quake.protocol*), 66
UpdateFrags (*class in vgio.quake.protocol*), 59
UpdateName (*class in vgio.quake.protocol*), 58
UpdateStat (*class in vgio.quake.protocol*), 55
uv (*vgio.devildaggers.hxmesh.Vertex attribute*), 8

## V

value (*vgio.quake.protocol.UpdateStat attribute*), 55
value (*vgio.quake2.bsp.TextureInfo attribute*), 77
value (*vgio.quake2.wal.Wal attribute*), 95
velocity (*vgio.quake.protocol.ClientData attribute*), 59
Version (*class in vgio.quake.protocol*), 55
version (*vgio.duke3d.map.Map attribute*), 20
version (*vgio.hexen2.bsp.Bsp attribute*), 25
version (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
version (*vgio.quake.bsp.bsp29a.Bsp attribute*), 38
version (*vgio.quake.mdl.Mdl attribute*), 46
version (*vgio.quake.spr.Spr attribute*), 67
version (*vgio.quake2.bsp.Bsp attribute*), 73
version (*vgio.quake2.md2.Md2 attribute*), 81
version (*vgio.quake2.sp2.Sp2 attribute*), 93
Vertex (*class in vgio.devildaggers.hxmesh*), 8
Vertex (*class in vgio.quake.bsp.bsp29*), 34
Vertex (*class in vgio.quake2.bsp*), 75
vertex_shader (*vgio.devildaggers.hxshader.HxShader attribute*), 12
vertexes (*vgio.hexen2.bsp.Bsp attribute*), 25
vertexes (*vgio.quake.bsp.bsp29.Bsp attribute*), 31

vertexes (*vgio.quake.bsp.bsp29.Edge attribute*), 37
vertexes (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
vertexes (*vgio.quake.mdl.Frame attribute*), 50
vertexes (*vgio.quake.mdl.Triangle attribute*), 49
vertexes (*vgio.quake2.bsp.Bsp attribute*), 73
vertexes (*vgio.quake2.bsp.Edge attribute*), 78
vertexes (*vgio.quake2.md2.Frame attribute*), 84
vertexes (*vgio.quake2.md2.Triangle attribute*), 83
vertices (*vgio.devildaggers.hxmesh.HxMesh attribute*), 7
vgio._core
    module, 96
vgio._core.__init__
    module, 96
vgio.devildaggers.hxmesh
    module, 7
vgio.devildaggers.hxresourcegroup
    module, 9
vgio.devildaggers.hxshader
    module, 12
vgio.devildaggers.hxtexture
    module, 13
vgio.duke3d
    module, 14
vgio.duke3d.art
    module, 14
vgio.duke3d.grp
    module, 16
vgio.duke3d.map
    module, 19
vgio.hexen2
    module, 25
vgio.hexen2.bsp
    module, 25
vgio.hrot
    module, 27
vgio.hrot.pak
    module, 28
vgio.quake
    module, 30
vgio.quake.bsp
    module, 30
vgio.quake.bsp.bsp29
    module, 30
vgio.quake.bsp.bsp29a
    module, 38
vgio.quake.lmp
    module, 43
vgio.quake.mdl
    module, 45
vgio.quake.pak
    module, 51
vgio.quake.protocol
    module, 54
vgio.quake.spr
    module, 67
vgio.quake.wad
    module, 69
vgio.quake2
    module, 72
vgio.quake2.bsp
    module, 72
vgio.quake2.dm2
    module, 80
vgio.quake2.md2
    module, 80
vgio.quake2.pak
    module, 84
vgio.quake2.protocol
    module, 87
vgio.quake2.sp2
    module, 93
vgio.quake2.wal
    module, 94
view_angles (*vgio.quake.protocol.MessageBlock attribute*), 67
view_height (*vgio.quake.protocol.ClientData attribute*), 59
visibilities (*vgio.hexen2.bsp.Bsp attribute*), 25
visibilities (*vgio.quake.bsp.bsp29.Bsp attribute*), 31
visibilities (*vgio.quake.bsp.bsp29a.Bsp attribute*), 39
visibilities (*vgio.quake2.bsp.Bsp attribute*), 73
visibilitiy_offset (*vgio.quake.bsp.bsp29.Leaf attribute*), 36
visibility (*vgio.duke3d.map.Sector attribute*), 22
visleafs (*vgio.hexen2.bsp.Model attribute*), 27
visleafs (*vgio.quake.bsp.bsp29.Model attribute*), 38
visleafs (*vgio.quake2.bsp.Model attribute*), 79
volume (*vgio.quake.protocol.Sound attribute*), 56
volume (*vgio.quake.protocol.SpawnStaticSound attribute*), 64
volume (*vgio.quake2.protocol.Sound attribute*), 89

## W

WadFile (*class in vgio.quake.wad*), 70
WadInfo (*class in vgio.quake.wad*), 72
Wal (*class in vgio.quake2.wal*), 95
Wall (*class in vgio.duke3d.map*), 24
wall_number (*vgio.duke3d.map.Sector attribute*), 21
wall_pointer (*vgio.duke3d.map.Sector attribute*), 21
walls (*vgio.duke3d.map.Map attribute*), 20
weapon (*vgio.quake.protocol.ClientData attribute*), 59
weapon (*vgio.quake2.protocol.MuzzleFlash attribute*), 87
weapon_frame (*vgio.quake.protocol.ClientData attribute*), 59
width (*vgio.devildaggers.hxtexture.HxTexture attribute*), 13

width (*vgio.quake.bsp.bsp29.Miptexture attribute*), 33
width (*vgio.quake.lmp.Lmp attribute*), 44
width (*vgio.quake.spr.Spr attribute*), 67
width (*vgio.quake.spr.SpriteFrame attribute*), 69
width (*vgio.quake2.sp2.SpriteFrame attribute*), 94
width (*vgio.quake2.wal.Wal attribute*), 95
write() (*vgio._core.__init__.ArchiveFile method*), 98
write() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 10
write() (*vgio.duke3d.art.ArtFile method*), 15
write() (*vgio.duke3d.grp.GrpFile method*), 18
write() (*vgio.duke3d.map.Sector class method*), 22
write() (*vgio.duke3d.map.Sprite class method*), 23
write() (*vgio.duke3d.map.Wall class method*), 24
write() (*vgio.hexen2.bsp.Model class method*), 27
write() (*vgio.hrot.pak.PakFile method*), 29
write() (*vgio.quake.bsp.bsp29.ClipNode class method*), 36
write() (*vgio.quake.bsp.bsp29.Edge class method*), 37
write() (*vgio.quake.bsp.bsp29.Face class method*), 36
write() (*vgio.quake.bsp.bsp29.Leaf class method*), 37
write() (*vgio.quake.bsp.bsp29.Miptexture class method*), 34
write() (*vgio.quake.bsp.bsp29.Model class method*), 38
write() (*vgio.quake.bsp.bsp29.Node class method*), 35
write() (*vgio.quake.bsp.bsp29.Plane class method*), 33
write() (*vgio.quake.bsp.bsp29.TextureInfo class method*), 35
write() (*vgio.quake.bsp.bsp29.Vertex class method*), 34
write() (*vgio.quake.bsp.bsp29a.ClipNode class method*), 41
write() (*vgio.quake.bsp.bsp29a.Edge class method*), 41
write() (*vgio.quake.bsp.bsp29a.Face class method*), 40
write() (*vgio.quake.bsp.bsp29a.Leaf class method*), 41
write() (*vgio.quake.bsp.bsp29a.Node class method*), 40
write() (*vgio.quake.mdl.Frame static method*), 51
write() (*vgio.quake.mdl.FrameGroup static method*), 51
write() (*vgio.quake.mdl.Skin static method*), 48
write() (*vgio.quake.mdl.SkinGroup static method*), 48
write() (*vgio.quake.mdl.StVertex class method*), 49
write() (*vgio.quake.mdl.Triangle class method*), 49
write() (*vgio.quake.mdl.TriVertex class method*), 50
write() (*vgio.quake.pak.PakFile method*), 52
write() (*vgio.quake.protocol.Bad static method*), 54
write() (*vgio.quake.protocol.CdTrack static method*), 65
write() (*vgio.quake.protocol.CenterPrint static method*), 63
write() (*vgio.quake.protocol.ClientData static method*), 60
write() (*vgio.quake.protocol.CutScene static method*), 66
write() (*vgio.quake.protocol.Damage static method*), 61
write() (*vgio.quake.protocol.Disconnect static method*), 55
write() (*vgio.quake.protocol.Finale static method*), 65
write() (*vgio.quake.protocol.FoundSecret static method*), 64
write() (*vgio.quake.protocol.Intermission static method*), 65
write() (*vgio.quake.protocol.KilledMonster static method*), 64
write() (*vgio.quake.protocol.LightStyle static method*), 58
write() (*vgio.quake.protocol.MessageBlock static method*), 67
write() (*vgio.quake.protocol.Nop static method*), 54
write() (*vgio.quake.protocol.Particle static method*), 61
write() (*vgio.quake.protocol.Print static method*), 57
write() (*vgio.quake.protocol.SellScreen static method*), 65
write() (*vgio.quake.protocol.ServerInfo static method*), 58
write() (*vgio.quake.protocol.SetAngle static method*), 57
write() (*vgio.quake.protocol.SetPause static method*), 63
write() (*vgio.quake.protocol.SetView static method*), 55
write() (*vgio.quake.protocol.SignOnNum static method*), 63
write() (*vgio.quake.protocol.Sound static method*), 56
write() (*vgio.quake.protocol.SpawnBaseline static method*), 62
write() (*vgio.quake.protocol.SpawnBinary static method*), 62
write() (*vgio.quake.protocol.SpawnStatic static method*), 62
write() (*vgio.quake.protocol.SpawnStaticSound static method*), 64
write() (*vgio.quake.protocol.StopSound static method*), 60
write() (*vgio.quake.protocol.StuffText static method*), 57
write() (*vgio.quake.protocol.TempEntity static method*), 63
write() (*vgio.quake.protocol.Time static method*), 56
write() (*vgio.quake.protocol.UpdateColors static method*), 60
write() (*vgio.quake.protocol.UpdateEntity static method*), 66
write() (*vgio.quake.protocol.UpdateFrags static method*), 59
write() (*vgio.quake.protocol.UpdateName static method*), 58
write() (*vgio.quake.protocol.UpdateStat static method*), 55
write() (*vgio.quake.protocol.Version static method*), 55
write() (*vgio.quake.spr.SpriteFrame static method*), 69
write() (*vgio.quake.spr.SpriteGroup static method*), 69

write() (*vgio.quake.wad.WadFile method*), 70
write() (*vgio.quake2.bsp.Area class method*), 80
write() (*vgio.quake2.bsp.AreaPortal class method*), 80
write() (*vgio.quake2.bsp.Brush class method*), 79
write() (*vgio.quake2.bsp.BrushSide class method*), 79
write() (*vgio.quake2.bsp.Edge class method*), 78
write() (*vgio.quake2.bsp.Face class method*), 77
write() (*vgio.quake2.bsp.Leaf class method*), 78
write() (*vgio.quake2.bsp.Lump class method*), 75
write() (*vgio.quake2.bsp.Model class method*), 79
write() (*vgio.quake2.bsp.Node class method*), 76
write() (*vgio.quake2.bsp.Plane class method*), 75
write() (*vgio.quake2.bsp.TextureInfo class method*), 77
write() (*vgio.quake2.bsp.Vertex class method*), 75
write() (*vgio.quake2.md2.Frame class method*), 84
write() (*vgio.quake2.md2.GlCommand class method*), 84
write() (*vgio.quake2.md2.GlVertex class method*), 84
write() (*vgio.quake2.md2.Skin class method*), 82
write() (*vgio.quake2.md2.StVertex class method*), 83
write() (*vgio.quake2.md2.Triangle class method*), 83
write() (*vgio.quake2.md2.TriVertex class method*), 83
write() (*vgio.quake2.pak.PakFile method*), 85
write() (*vgio.quake2.protocol.Bad class method*), 87
write() (*vgio.quake2.protocol.CenterPrint class method*), 91
write() (*vgio.quake2.protocol.ConfigString class method*), 91
write() (*vgio.quake2.protocol.DeltaPacketEntities class method*), 92
write() (*vgio.quake2.protocol.Disconnect class method*), 89
write() (*vgio.quake2.protocol.Download class method*), 92
write() (*vgio.quake2.protocol.Frame class method*), 93
write() (*vgio.quake2.protocol.Inventory class method*), 88
write() (*vgio.quake2.protocol.Layout class method*), 88
write() (*vgio.quake2.protocol.MessageBlock static method*), 93
write() (*vgio.quake2.protocol.MuzzleFlash class method*), 87
write() (*vgio.quake2.protocol.MuzzleFlash2 class method*), 88
write() (*vgio.quake2.protocol.Nop class method*), 89
write() (*vgio.quake2.protocol.PacketEntities class method*), 92
write() (*vgio.quake2.protocol.PlayerInfo class method*), 92
write() (*vgio.quake2.protocol.Print class method*), 90
write() (*vgio.quake2.protocol.Reconnect class method*), 89
write() (*vgio.quake2.protocol.ServerData class method*), 91
write() (*vgio.quake2.protocol.Sound class method*), 90
write() (*vgio.quake2.protocol.SpawnBaseline class method*), 91
write() (*vgio.quake2.protocol.StuffText class method*), 90
write() (*vgio.quake2.protocol.TempEntity class method*), 88
writestr() (*vgio._core.__init__.ArchiveFile method*), 98
writestr() (*vgio.devildaggers.hxresourcegroup.HxResourceGroupFile method*), 10
writestr() (*vgio.duke3d.art.ArtFile method*), 15
writestr() (*vgio.duke3d.grp.GrpFile method*), 18
writestr() (*vgio.hrot.pak.PakFile method*), 29
writestr() (*vgio.quake.pak.PakFile method*), 53
writestr() (*vgio.quake.wad.WadFile method*), 71
writestr() (*vgio.quake2.pak.PakFile method*), 86

## X

x (*vgio.duke3d.map.Sprite attribute*), 22
x (*vgio.duke3d.map.Wall attribute*), 24
x (*vgio.quake.bsp.bsp29.Vertex attribute*), 34
x (*vgio.quake.mdl.TriVertex attribute*), 50
x (*vgio.quake2.bsp.Vertex attribute*), 75
x (*vgio.quake2.md2.TriVertex attribute*), 82
x_offset (*vgio.duke3d.map.Sprite attribute*), 23
x_panning (*vgio.duke3d.map.Wall attribute*), 24
x_repeat (*vgio.duke3d.map.Sprite attribute*), 23
x_repeat (*vgio.duke3d.map.Wall attribute*), 24
x_velocity (*vgio.duke3d.map.Sprite attribute*), 23

## Y

y (*vgio.duke3d.map.Sprite attribute*), 22
y (*vgio.duke3d.map.Wall attribute*), 24
y (*vgio.quake.bsp.bsp29.Vertex attribute*), 34
y (*vgio.quake.mdl.TriVertex attribute*), 50
y (*vgio.quake2.bsp.Vertex attribute*), 75
y (*vgio.quake2.md2.TriVertex attribute*), 82
y_offset (*vgio.duke3d.map.Sprite attribute*), 23
y_panning (*vgio.duke3d.map.Wall attribute*), 24
y_repeat (*vgio.duke3d.map.Sprite attribute*), 23
y_repeat (*vgio.duke3d.map.Wall attribute*), 24
y_velocity (*vgio.duke3d.map.Sprite attribute*), 23

## Z

z (*vgio.duke3d.map.Sprite attribute*), 22
z (*vgio.quake.bsp.bsp29.Vertex attribute*), 34
z (*vgio.quake.mdl.TriVertex attribute*), 50
z (*vgio.quake2.bsp.Vertex attribute*), 75
z (*vgio.quake2.md2.TriVertex attribute*), 82
z_velocity (*vgio.duke3d.map.Sprite attribute*), 23