Scripting Plugin for Wings3D

Author: Edward Blake

April 7, 2023

This document is split in three sections:

1. Script Information (.wscr)
2. Scheme Scripting
3. Python Scripting

Script Information (.wscr)

What are .wscr files?

Wings SCRipt files provide information so that scripts can be used like normal plugins in wings, WSCR files specify:

- Name and description of the script.
- If the script is a import, export, shape or command script.
- Parameters to ask the user before starting the script.

When a user opens a script chooser dialog, the script search path is traversed for all wscr files, the name in the wscr file is used for the name shown in the script chooser dialog.

`type` Directive

```
type "py"
```

This is either “scm” or “py”, depending on if the script is a Scheme or Python file.

`name` Directive

```
name ?__(1,"Raw Triangles (.raw)")
```

A name for the script, which appears in the script chooser dialog.

`mode` Directive

```
mode "import"
```

The mode indicates what kind of script it is. The mode determines in which script chooser the script will show up. If a script has "export" for mode, it will only show up when choosing "Script-based exporters" in the "File" "Export" menu.

Kind of script	mode
New shapes	
Importers	import
Exporters	export
Simple whole object commands	simple_body_command

New shapes scripts create a new shape.

Importer scripts are Found in File > Import. Returns an E3D file

Exporter scripts are Found in File > Export. Takes an E3D file.

Simple whole object command scripts changes point coordinates and vertex attributes.

`desc` Directive

```
desc ?__(2,"This script does something")
```

A short description can be added to provide more information about what the script does.

`include` Directives

```
include "(%)_dialog.inclr"
```

WSCR files can include other files, it is recommended to move all the parameter related configuration to another file and add an include directive towards it so as to keep the main wscr file small. This is important as each time the user opens a script chooser dialog, every directory is traversed for every wscr file, and

each wscr file found is read for it's information to display in the script chooser browser.

The script chooser traversal process reads every wscr file, but it does not read the include directives. When a script is invoked, the include directives are used to read the rest of the configuration. A parenthesis wrapped percent symbol "(%)" auto expands to the name of the .wscr file before the extension.

`params_title` Directive

```
params_title ?__(3,"Command Settings")
```

The params_title directive indicates the titlebar string of the dialog that is shown for user input.

`params_templates` Section

```
params_templates {
  template "export" ""
}
```

Param templates are standard groups of parameters that are used by most plugins in Wings3D for consistency. For example, importers might use the import template to show the flip axis and scale parameters. When a template is used, the corresponding code that interprets the standard parameters to automatically modify the E3D tuple will also be handled by the scripting interface.

`template` Directive

```
template "export" ""
```

A template in the params_template section.

`params` Section

```
params {
  param "Size" "2.0"
  ...
}
```

The params section contains param directives.

`param` Directive

```
param "Size" "2.0"
```

A param directive specifies one field for user input

`extensions` Section

```
extensions {
  ext ".tres" ?__(5,"Godot Mesh Library")
}
```

The extensions section indicates the extensions that are supported by the importer or exporter script. Each line within the section should be a ext directive, with its first argument the extension including the dot, and the second argument is a description of the file format.

`params_init` Instructions Section

```
params_init {
  do "1>$'subdivisions'"
}
```

The params_init section behaves as a subroutine, with each directive being evaluated sequentially. This section is used to load values into temporary variables to pass to the params and params_set section. The state of temporary variables do not carry over after the params section. The argument of a `do` instruction is not a simple string. Refer to **The Query Mini Language** in the next section on how to write the argument.

`params_set` Instructions Section

```
params_set {
  export_param "include_uvs"  "bool(params/include_uvs)"
  ...
}
```

The params_set section behaves as a subroutine, with each directive within assigning a value to a parameter sequentially. This detail is relevant as the variable query string of the instructions can set temporary variables for later instructions to use.

`export_param` Instruction

```
export_param "subdivisions" "params/subdivisions"
```

Sets an export parameter for the `wpa:export` function or some scripting export functions on the scripting plugin side. For "`script_texture_convert`", the recommended option is "`'user'`" with the export template for the texture handling. The second argument is not a simple string. Refer to **The Query Mini Language** in the next section on how to write the second argument.

`import_param` Instruction

```
import_param "script_texture_convert" "'auto'"
```

Sets an import parameter for the `wpa:import` function or some scripting import functions on the scripting plugin side. For "`script_texture_convert`", the recommended option is "`'auto'`" for the texture handling. The second argument is not a simple string. Refer to **The Query Mini Language** in the next section on how to write the second argument.

`script_param` Instruction

```
script_param "test" "1"
```

Sets an extra parameter for the script, which has to be accessed through the `extra_params` variable in Python, or the `*extra-params*` variable in Scheme. The second argument is not a simple string. Refer to **The Query Mini Language** in the next section on how to write the second argument.

`extra_file` Section

```
extra_file "icon" {
  title "Choose small image"
  extensions {
    ...
  }
}
```

An extra input file which can be used by any of the script modes. The first argument should be the name that will be set in `extra_params` for the script. Set the title and extensions for it.

Language files

Like normal wings3d plugins, wscr files can use language files to localize the user interface text into multiple languages. The lang file needs to be in the same directory as the wscr file, and begin with the same base file name without the extension, with a underscore and a language code following it, followed by the ".lang" file extension. The format of the language file is the same Erlang term format as wings3d lang files. To localize a string in the wscr file, enclose the string in the same enclosing syntax used in Erlang wings3d source files.

```
name ?__(1,"Color")
```

For a script called "script_name.wscr", the lang file for French should be named "script_name_fr.lang", and the content resembles other Wings3D lang files but the outer name is `script` and the inner name is `wscr`:

```
{script,
 [
  {wscr,
   [
    {1,"Couleur"}
   ]}
 ]}.
```

The Query Mini Language

The query language is designed to be able to navigate the Erlang terms used by Wings3D with a simple path-like syntax.

When the user enters parameter settings, before the script is invoked, there will be a need to set some elaborate settings for `wpa:export` or `wpa:import` through `export_param` and `import_param` instructions. The query language is used for referencing parameter values in the script information file as well as for communicating between the script and plugin.

First Identifier

The first identifier in the query can be an identifier such as st, params, a temporary variable retrieval, a function call, a tuple/list constructor or a literal such as 1, 'atom' or "string".

Atoms

Atoms must be enclosed in single quotes, words without single quotes are not automatically made into atoms.

e.g.

```
'triangulated'
```

Strings

Strings are enclosed in double quotes.

e.g.

```
"Sample text"
```

Numbers

Integers and floating point numbers can be used in the query.

e.g.

```
1
1.2
```

Dot Notation

```
variable.shapes
```

With some records such as 'st' and 'we', fields of those records can be accessed using a dot notation. There is no need to specify the record type itself as done in Erlang because the record is inspected at runtime to match the record type for a field name.

However, only a few record types have their fields registered in the plugin to use the dot notation. Alternatively, record fields can be accessed with the tuple index operator, where {0} is the atom of the record and {1} is the first field.

e.g.

```
st.shapes
```

Associative List Path

```
variable/key
```

Associative lists can be navigated by their keys with the "slash" (/) character. If the associated lists contain more associated lists, the slash can be chained like a path to access deeper elements.

e.g.

```
params/include_uvs
```

Tuple by index

```
variable{0}
```

The entries of a tuple can be accessed by a zero-based index by enclosing a number inside curly brackets, the first entry is {0}. Indexing operators can be chained as necessary to access deeper elements.

e.g.

```
tuple(1,2,3,4){2}
```

List by index

```
variable[0]
```

The entries of a list can be accessed by a zero-based index by enclosing a number inside square brackets, the first entry is [0] Indexing operators can be chained as necessary to access deeper elements.

e.g.

```
list(1,2,3,4)[1]
```

Store to

```
>$'variable'
```

Store the current value into a temporary variable for later retrieval, the syntax is a greater than symbol, a dollar sign and a integer or atom literal. This operator returns the same value that it is storing so it can be added at the end of a query chain without affecting the return value.

e.g.

```
params/include_uvs>$'include_uvs'
```

Get from

```
<$'variable'
```

Get the value stored in a temporary variable, the syntax is a lesser than symbol, a dollar sign and a integer or atom literal, this can appear at the beginning of the query.

e.g.

```
<$'settings'/group1/setting
```

Call function

```
module:function( ... )
```

Calls a function from a module, with the given arguments. A function call can appear at the beginning of the query.

e.g.

```
lists:reverse(list(1,2,3,4))
```

List constructor

```
list(1,'two',3)
```

Construct a list with given values. A list constructor can appear at the beginning of the query. It is usually more convenient to use %setvar with temporary variables to input list data structures from the script to the plugin than to use inline constructors.

Tuple constructor

```
tuple(1,2,'three')
```

Construct a tuple with given values. A tuple constructor can appear at the beginning of the query. It is usually more convenient to use %setvar with temporary variables to input tuple data structures from the script to the plugin than to use inline constructors.

Boolean Value

```
bool(1)
```

Get either the atom 'true' or 'false' from another value.

Integer Value

```
int("1")
```

Get the integer value from another value

Float Value

```
float("1.0")
```

Get the float value from another value.

Ok Test

```
ok_test(orddict:find(key, $<'variable'), "not found")
```

Tests if the return of the first argument is a tuple with the first element being 'ok'. If it is an 'ok' tuple, the tuple's value is returned. If the first argument is not an 'ok' tuple, the second argument is evaluated and returned instead.

Value Test

```
value_test('value', gb_trees:lookup(key, $<'variable'), "not found")
```

A general version of ok_test that tests if the return of the second argument is a tuple with the first element being equal to the first argument. If the second argument does not match, the third argument is evaluated and returned instead.

Conditional If

```
if(bool(<$'cond'), "on", "off")
```

Tests the first argument if it evaluates to the atom 'true', then the second argument (the "then" argument) is evaluated, otherwise the third (the "else" argument) is evaluated.

Scheme Scripting Guide for Wings3D

Online Resources for the Scheme Language

A very non exhaustive list of resources:
Yet Another Scheme Tutorial
http://www.shido.info/lisp/idx_scm_e.html
The Scheme Programming Language 4th ed.
https://www.scheme.com/tspl4/
Simply Scheme: Introducing Computer Science
https://people.eecs.berkeley.edu/~bh/ss-toc2.html
R5RS (reference for functions found in nearly every Scheme)
https://conservatory.scheme.org/schemers/Documents/Standards/R5RS/
Gauche reference
http://practical-scheme.net/gauche/man/gauche-refe/index.html

.wscr File

For your script to be able to appear in the script chooser dialog, you will need to make a .wscr file with the same base name as your main Scheme script file. For example, if your script file is example.scm, your wscr file should be named example.wscr. The name that appears in the script chooser is the name specified in the .wscr file.

Details Of The Script

Initial Bootstrap Script

A bootstrap script that comes with the script plugin is first launched which then defines some basic functions, sometimes adds include paths, and gets from stdin the actual script being invoked as well as parameters that goes with it.

Parameters that are passed in

When a script is invoked, parameters from the dialog box are passed in, other parameters are also passed in depending on the type of script.

Returning values from the script

Scripts write to stdout their response before exiting.

```
(list 'ok '(new_shape ... ))
```

Calling functions in Wings3D from a script

Scripts can communicate with the plugin by setting temporary variables and using queries to call functions and get the return value. Temporary variables makes it easier to input elaborate data structures as arguments by referencing them in the query.

Set value to variable:

```
(list '%setvar "var1" '(1 2 3 4))
```

The reply sent back for setvar:

```
(list '%setok "var1")
```

Query:

```
(list '%query "lists:reverse(<$'var1')")
```

The reply sent back for queries:

```
(list '%response '(4 3 2 1))
```

New Shape Scripts

New shape scripts are available when right clicking with no selection

An example of a complete Scheme script to create a shape and send it back to Wings3D:

```
(write (list 'new_shape "MyShape"
        '(
                (0 1 3 2)
                (6 7 5 4)
                (1 0 4 5)
                (2 3 7 6)
                (0 2 6 4)
                (3 1 5 7)
        ) '(
                #(-2.0 0.5 -2.0)
                #(-2.0 0.5 2.0)
                #(2.0 0.5 -2.0)
                #(2.0 0.5 2.0)
                #(-2.0 -0.5 -2.0)
                #(-2.0 -0.5 2.0)
                #(2.0 -0.5 -2.0)
                #(2.0 -0.5 2.0)
        )))
```

The e3d object way is also possible, all "new shape" scripts write a list where the first element is the symbol new_shape:

```
(define Fs ...) ; List of indices into vertice list
(define Vs ...) ; Vertice list
(write (list 'new_shape "MyShape" Fs Vs))
```

Command Scripts

Command scripts are available when right clicking on a selected object

Return set_points

The "points" extra parameter is needed to access the list of vertices, which come as a vertice number and a 3 dimensional position.

```
(define op     (list-ref
                 (assoc "op" *extra-params*) 1))
(define points (list-ref
                 (assoc "points" *extra-params*) 1))
```

The following script shifts all the vertice positions of a selected object by 0.5:

```
(define op      (list-ref
                  (assoc "op"      *extra-params*) 1))
(define points (list-ref
                  (assoc "points" *extra-params*) 1))
(define newpoints
        (map (lambda (p)
                (define a (vector-ref p 0))
                (define b (vector-ref p 1))
                (define x (vector-ref b 0))
                (define y (vector-ref b 1))
                (define z (vector-ref b 2))
                (vector a (vector
                    (+ 0.5 x)
                    (+ 0.5 y)
                    (+ 0.5 z)))
        ) points))
(newline)
(write (list (list 'set_points newpoints )))
```

Importer Scripts

Importer scripts are available from the "File" > "Import" menu.

Return e3d_file

The "filename" extra parameter is needed which contain the file path of the file to read:

```
(define filename
  (list-ref
    (assoc "filename" *extra-params*) 1))
```

Constructing e3d objects involves using make-e3d_face, make-e3d_object, make-e3dfile, etc.

After parsing the file, the importer writes the contents of e3d_file back to Wings3D:

```
(newline)
(write (list 'ok (make-e3d_file `(objs ,(list Obj)))))
```

Exporter Scripts

Exporter scripts are available from the "File" > "Export" and "File" > "Export Selected" menus.

Read e3d_file

The "filename" and "content" extra parameters are needed, the file name is chosen by the user to save the model, and the content contains an e3d_file data structure.

```
(define filename
  (list-ref
    (assoc "filename" *extra-params*) 1))
(define content
  (list-ref
    (assoc "content"  *extra-params*) 1))
```

After writing the content to a file, the exporter writes an ok to Wings3D:

```
(newline)
(write '(ok))
```

E3D Function Reference

The E3D helper functions are loaded already when using Scheme.

e3d_face

(**make-e3d_face** . sc)
Make a e3d_face list structure with a constructor list, available items are 'vs 'vc 'tx 'ns 'mat 'sg 'vis
(**e3d_face?** l)
Is the list structure an e3d_face?
(**e3d_face-vs** l)
(**e3d_face-vc** l)
(**e3d_face-tx** l)
(**e3d_face-ns** l)
(**e3d_face-mat** l)
(**e3d_face-sg** l)
(**e3d_face-vis** l)
Get the value of a given field in e3d_face.

e3d_mesh

(**make-e3d_mesh** . sc)
Make a e3d_mesh list structure with a constructor list, available items are 'type 'vs 'vc 'tx 'ns 'fs 'he 'matrix
(**e3d_mesh?** l)
Is the list structure an e3d_mesh?
(**e3d_mesh-type** l)
(**e3d_mesh-vs** l)

(**e3d_mesh-vc** l)
(**e3d_mesh-tx** l)
(**e3d_mesh-ns** l)
(**e3d_mesh-fs** l)
(**e3d_mesh-he** l)
(**e3d_mesh-matrix** l)
Get the value of a given field in e3d_mesh

e3d_object

(**make-e3d_object** . sc)
Make a e3d_object list structure with a constructor list, available items are 'name 'obj 'mat 'attr
(**e3d_object?** l)
Is the list structure an e3d_object?
(**e3d_object-name** l)
(**e3d_object-obj** l)
(**e3d_object-mat** l)
(**e3d_object-attr** l)
Get the value of a given field in e3d_object

e3d_file

(**make-e3d_file** . sc)
Make a e3d_file list structure with a constructor list, available items are 'objs 'mat 'creator 'dir
(**e3d_file?** l)
Is the list structure an e3d_file?
(**e3d_file-objs** l)
(**e3d_file-mat** l)
(**e3d_file-creator** l)
(**e3d_file-dir** l)
Get the value of a given field in e3d_file

e3d_image

(**make-e3d_image** . sc)
Make a e3d_image list structure with a constructor list, available items are 'type 'bytes_pp 'alignment 'order 'width 'height 'image 'filename 'name 'extra
(**e3d_image?** l)
Is the list structure an e3d_image?
(**e3d_image-type** l)
(**e3d_image-bytes_pp** l)
(**e3d_image-alignment** l)
(**e3d_image-order** l)
(**e3d_image-width** l)

(**e3d_image-height** l)
(**e3d_image-image** l)
(**e3d_image-filename** l)
(**e3d_image-name** l)
(**e3d_image-extra** l)
Get the value of a given field in e3d_image

Python Scripting Guide for Wings3D

Online Resources for the Python language

A non exhaustive list of resources:
Python Tutorial
https://docs.python.org/3/tutorial/
Python Language Reference
https://docs.python.org/3/reference/

.wscr File

For your script to be able to appear in the script chooser dialog, you will need to make a wscr file with the same base name as your main python script file. For example, if your script file is example.py, your wscr file should be named example.wscr. The name that appears in the script chooser is the name specified in the wscr file.

Details Of The Script

Initial Bootstrap Script

A bootstrap script that comes with the script plugin is first launched which then defines some basic functions, sometimes adds include paths, and gets from stdin the actual script being invoked as well as parameters that goes with it.

Parameters that are passed in

When a script is invoked, parameters from the dialog box are passed in, other parameters are also passed in depending on the type of script.

Returning values from the script

Scripts write to stdout an OutputList before exiting.

Calling functions in Wings3D from a script (WIP)

Scripts can communicate with the plugin by setting temporary variables and using queries to call functions and get the return value. Temporary variables makes it easier to input elaborate data structures as arguments by referencing them in the query.

(WIP)

Set value to variable:

```
(list '%setvar "var1" '(1 2 3 4))
```

The reply sent back for setvar:

```
(list '%setok "var1")
```

Query:

```
(list '%query "lists:reverse(<$'var1')")
```

The reply sent back for queries:

```
(list '%response '(4 3 2 1))
```

New Shape Scripts

New shape scripts are available when right clicking with no selection.

A complete example of a Python script to create a new shape:

```
b = w3d_newshape.NewShape()
b.fs = w3d_newshape.ListOfArrays([
          [0,1,3,2],
          [6,7,5,4],
          [1,0,4,5],
          [2,3,7,6],
          [0,2,6,4],
          [3,1,5,7]
])
b.vs = w3d_newshape.ListOfTuples([
          (-2.0,0.5,-2.0),
          (-2.0,0.5,2.0),
          (2.0,0.5,-2.0),
          (2.0,0.5,2.0),
          (-2.0,-0.5,-2.0),
          (-2.0,-0.5,2.0),
          (2.0,-0.5,-2.0),
          (2.0,-0.5,2.0)
])
o = b.as_output_list()
o.write_list_out(sys.stdout)
```

To create a shape, create a geometry with E3DMesh and E3DObject, and assign E3DObject to the obj attribute of a NewShape object. And then output the output list of NewShape to stdout.

```
e3d_o = w3d_e3d.E3DObject()
e3d_o.obj = mesh
nshp = w3d_newshape.NewShape()
nshp.prefix = "shape"
nshp.obj = e3d_o
o = nshp.as_output_list()
o.write_list_out(sys.stdout)
```

Command Scripts

Command scripts are available when right clicking on a selected object.

Return set_points

(WIP)

The "points" extra parameter is needed to access the list of vertices, which come as a vertice number and a 3 dimensional position.

```
points = extra_params["points"]
```

Importer Scripts

Importer scripts are available from the "File" > "Import" menu.

The "filename" extra parameter is needed which contain the file path of the file to read:

```
filename = extra_params["filename"]
```

Importers take the filename to read the file needed, construct a E3DObject, add it to an E3DFile and return the output list as part of an "ok" tuple.

Constructing E3D objects involves instancing E3DFace, E3DObject, E3DFile, etc.

```
print("")
e3df = w3d_e3d.E3DFile()
e3df.objs = [obj] # obj is an E3DObject
o_ok = OutputList()
o_ok.add_symbol("ok")
o_ok.add_list(e3df.as_output_list())
o_ok.write_list_out(sys.stdout)
```

Exporter Scripts

Exporter scripts are available from the "File" > "Export" and "File" > "Export Selected" menus.

Read E3DFile

The "filename" and "content" extra parameters are needed, the file name is chosen by the user to save the model, and the content contains an e3d_file data structure.

```
filename = extra_params["filename"]
content = w3d_e3d.E3DFile()
content.load_from(extra_params["content"])
```

Exporters take in both a filename and the e3d_file content which has to be loaded with E3DFile.load_from(...). After writing the file to disk, create an "ok" tuple and write its output list out:

```
print("")
o = OutputList()
o.add_symbol("ok")
o.write_list_out(sys.stdout)
```

OutputList Function Reference

Python is somewhat different from Erlang and Scheme for their data structures, so to serialize objects for input and output with the script plugin, a helper class called OutputList is used. Other helper classes also use OutputList to be able to load and write data to and from Wings3D.

class OutputList

Visible attributes lst_cont, lst_type

OutputList()

Create a new empty OutputList instance

list.**add_symbol**(a)

Add a symbol (interned string) to the list

list.**add_str**(a)

Add a string to the list

list.**add_number**(a)

Add a number to the list

list.**add_numbers**(alst)

Add several numbers to the list, it does not add a sublist

list.**add_integer**(a)

Add an integer to the list

list.**add_integers**(alst)

Add several integers to the list, it does not add a sublist

list.**add_float**(a)

Add a floating point number to the list

list.**add_floats**(alst)

Add several floating point numbers to the list, it does not add a sublist

list.**add_list**(a)

Add a sublist to the list

list.**add_vector**(a)

Add a vector (tuple) to the list

list.**add**(a, typ)

Add an item with type number to the list, this method shouldn't be used directly.

list.**write_list_out**(ost)

Write the list to the stream ost

NewShape (w3d_newshape) Function Reference

The w3d_newshape module contains NewShape helper classes and can be imported with:

```
import w3d_newshape
```

class ListOfArrays

Visible attributes: l

ListOfArrays()

Create a new empty ListOfArrays instance

list.**as_output_list**()

Return an OutputList instance

class ListOfTuples

Visible attributes: l

ListOfTuples()

Create a new empty ListOfTuples instance

list.**as_output_list**()

Return an OutputList instance

class NewShape

Visible attributes prefix, fs, vs, obj, mat

NewShape()

Create a NewShape instance

shape.**as_output_list**()

Return an OutputList instance, the kind of tuple that is returned is different depending on if obj is set, if it is set the tuple contains obj and mat. If it is not set, it returns a tuple with fs and vs.

```
e3d_o = w3d_e3d.E3DObject()
e3d_o.obj = mesh
shp = w3d_newshape.NewShape()
shp.prefix = "shape"
shp.obj = e3d_o
o = shp.as_output_list()
o.write_list_out(sys.stdout)
```

E3D (w3d_e3d) Function Reference

The w3d_e3d module contains E3D helper classes and can be imported with:

```
import w3d_e3d
```

class E3DFace

Visible attributes vs, vc, tx, ns, mat, sg, vis

E3DFace()

Creates a new E3DFace instance

face.**as_output_list**()

Outputs the whole E3DFace instance and all its values into an output list, not usually used directly.

face.**load_from**(expr)

Load contents into a E3DFace instance, not usually used directly.

class E3DMesh

Visible attributes: type, vs, vc, tx, ns, fs, he, matrix

E3DMesh()

Creates a new E3DMesh instance

mesh.**as_output_list**()

Outputs the whole E3DMesh instance and all its values into an output list, not usually used directly.

mesh.**load_from**(expr)

Load contents into a E3DMesh instance, not usually used directly.

class E3DObject

Visible attributes name, obj, mat, attr

E3DObject()
Creates a new E3DObject instance
obj.**as_output_list**()
Outputs the whole E3DObject instance and all its values into an output list, not usually used directly.
obj.**load_from**(expr)
Load contents into a E3DObject instance, not usually used directly.

class E3DImage

Visible attributes type, bytes_pp, alignment, order, width, height, image, filename, name, extra

E3DImage()
Creates a new E3DImage instance
image.**as_output_list**()
Outputs the whole E3DImage instance and all its values into an output list, not usually used directly.
image.**load_from**(expr)
Load contents into a E3DImage instance, not usually used directly.

class MaterialMaps

Visible attributes maps

MaterialMaps()
Create a new MaterialMaps instance
maps.**as_output_list**()
Outputs the whole MaterialMaps instance and all its values into an output list, not usually used directly.
maps.**load_from**(tlist)
Load contents into a MaterialMaps instance, not usually used directly.

class MaterialOpenGLAttributes

Visible attributes ambient, specular, shininess, diffuse, emission, metallic, roughness, vertex_colors

MaterialOpenGLAttributes()
Creates a new MaterialOpenGLAttributes instance
opengl.**as_output_list**()
Outputs the whole MaterialOpenGLAttributes instance and all its values into an output list, not usually used directly.
opengl.**load_from**(tlist)
Load contents into a MaterialOpenGLAttributes instance, not usually used directly.

class Material

Visible attributes name, attrs

Material()
Creates a new Material instance

mat.**as_output_list**()
Outputs the whole Material instance and all its values into an output list, not usually used directly.

mat.**load_from**(expr)
Load contents into a Material instance, not usually used directly.

class E3DFile

Visible attributes objs, mat, creator, dir

E3DFile()
Creates a new E3DFile instance

e3df.**as_output_list**()
Outputs the whole E3DFile instance and all its values into an output list which is then to be written to stdout

```
b = w3d_e3d.E3DFile()
b.objs.append(E3DObject())
o_ok = OutputList()
o_ok.add_symbol("ok")
o_ok.add_list(b.as_output_list())
o_ok.write_list_out(sys.stdout)
```

e3df.**load_from**(expr)
Load contents into a E3DFile instance

```
content = w3d_e3d.E3DFile()
content.load_from(extra_params["content"])
```