Guerilla parser

Release 0.8.3

Mar 10, 2022

Contents

1 Table of content **3**
- 1.1 Changelog . . . 3
- 1.2 Example . . . 6
- 1.3 Reference . . . 9
- 1.4 Known limitations . . . 13
- 1.5 Guerilla file format info . . . 14

Python Module Index **17**

Index **19**

Python Guerilla file parser is a python module aimed to parse and retrieve Guerilla render files content easily. It supports .gproject, .glayer, and other files written by Guerilla.

This module does not require any Python package beyond the standard ones.

If you don't know this module, the first thing to do is go the *Example* section to see various use cases.

CHAPTER 1

Table of content

1.1 Changelog

This changelog keep track of modifications. Keep an eye on it when changing versions. Some advices are often provided.

1.1.1 0.8.3 (2022 03 10)

- Add support for `types.multistrings` plug type.
- Leave the a `types.float` value to string if float parsing fail.
- Support spaces in plug names (`"$1.Min Trace Depth"`).

1.1.2 0.8.2 (2021 04 01)

- Add support for `types.colorspaces` plug type as `str`.

1.1.3 0.8.1 (2021 03 07)

- Add support for `types.animationmode` plug type as `str`.

1.1.4 0.8.0 (2021 03 06)

Officially support Python 3.7 and 3.8.

Add method `GuerillaParser.path_to_plug(path)` ` to find plug from its path.

Fix `GuerillaParser.plugs` property didn't return all plugs properly.

Fix crash when nodes are created with a negative zero `-0` as name.

Limit parser crash in some specific cases on corrupted file values enclosing regex: `0,5` (comma) no more match float regex.

- Uppercase "aov" in docstrings, documentation and exceptions.
- Cleanup .gitignore, add building folders.
- Update copyright date.
- Docstring and documentation cleanup.
- Add support for `types.radians0pi4` plug type as `float`.

1.1.5 0.7.0 (2020 06 09)

Fix `LUIPSTypeInt` with `'nil'` value now return `None`.

Fix new plug type parsing:

- `LUIPSTypeFloat01Open` as `float`.
- `types.materials` as `str`.
- `LUIPSTypeVector` as `tuple`.
- `LUIPSTypePoint` as `tuple`.
- `LUIPSTypeNumber` as `float`.
- `types.radians` as `float`.

Fix doc to add instance variable type.

Improve `ArchReference` node type support adding a `ReferenceFileName` plug to such node.

Fix some deprecation warning on Python 3.8.

`connect()` and `depend()` commands referencing document root attributes on `.glayer` and `.grendergraph` are skipped:

connect("$1.AspectRatio","$0.ProjectAspectRatio") depend("$17.Out","$0|Preferences.ShutterClose")

1.1.6 0.6.0 (2018 02 27)

Add Python 3.5 and 3.6 CI test. Now officially support 2.7, 3.4, 3.5 and 3.6.

`connect()` and `depend()` call with nodes having dots `.` is now supported.

Support `^` character in `set()` command.

New iterator: `GuerillaParser.plugs` iterate over every parsed `GuerillaPlug`.

New plug types found and supported:

- `HSetPlug`
- `HVisiblePlug`
- `HMattePlug`
- `SceneGraphNodePropsPlug`
- `SceneGraphNodeRenderPropsPlug`
- `AttributePlug`

- `AttributeShaderPlug`

See *this page* for more information.

Fix node with number as name (`Cube`, `Sphere`, `Plane`, etc. with type `SubPrimitive`). Guerilla return their name as number (`0` for default `Cube` sub primitive) but they are bracketed in paths `Cube|[0]`. Now, this behavior is properly emulated.

Fix `GuerillaPlug.path` could raise an exception if plug's parent is the root node.

Fix `GuerillaParser() diagnose` argument would crash when trying to print root node paths.

Fix no more printing of various unknown commands.

Fix `GuerillaParser.nodes` property wasn't iterating in every nodes.

Still improve documentation.

Rewrite most regex.

Handle Guerilla paths for numeric node names (`SubPrimitive` typed nodes).

1.1.7 0.5.0 (2018 02 24)

Add Python 3 support (and CI). Now officially support 2.7 and 3.4.

Support new characters for node names:

- brackets (`[]`).

Fix:

- Implicit node paths were not properly parsed.
- Guerilla file encoding is `iso-8859-1` (was broken in Python 3)

Unit tests: Improve performance parsing once and use later.

Documentation:

- Uppercase first letter of every docstring.
- Remove useless quotes from class names.
- Rewrite most of the documentation.

1.1.8 0.4.0 (2018 02 13)

Support new characters:

- slash (`/`) in path of `set()` commands.
- comma (`,`), dollar (`$`) and minus (`-`) in path of `connect()` commands.

Improve documentation formating.

Fix unit test in environment with default guerilla_parser module.

Reorganize unit tests.

1.1.9 0.3.0 (2018 01 10)

Support new characters for node names:

- parentheses (`()`).
- spaces, dot (`.`).
- backslash (`\\`).
- dollar (`$`).
- pipe (`|`).
- plus (+).
- minus (−).
- space.

Support nodes with empty names (`GraphFrame` can have empty string as name).

Fix bad implicit node handling (rewrite implementation).

Write a `__repr__()` implementation for `GuerillaNode` and `GuerillaPlug` for debugging purpose.

Rewrite unit test implementation to dynamically create them.

1.1.10 0.2.0 (2017 11 4)

Better performance on big gproject files.

Support nodes with , and | in names.

Fix bad assertion in plug name.

Skip unsupported (yet) inputs in *$0* formatting.

Update documentation (still far from perfect).

1.1.11 0.1.0 (2017 06 11)

Initial release

1.2 Example

This page give some examples and code snippets to use Guerilla parser.

1.2.1 Parse given file

Use the *`parse()`* method to parse and get *`GuerillaParser`* object interfacing the parsed document:

```
>>> import guerilla_parser
>>> p = guerilla_parser.parse("/home/user/my_project.gproject")
```

1.2.2 Iterate over every node

This snippet use *GuerillaParser.nodes* property to iterate over every parsed node.

```
>>> for node in p.nodes:
>>>     print node.path, node.type
```

1.2.3 Get every reference files

This snippet list every reference file.

This can be useful to list Alembic files from a gproject.

```
>>> for node in p.nodes:
>>>     if node.type == 'ArchReference':
>>>         print node.get_plug('ReferenceFileName')
```

1.2.4 Get root node

For a gproject file, root node is the document node.

```
>>> doc = p.root
>>> print doc.get_plug('FirstFrame').value
101
>>> print doc.get_plug('LastFrame').value
150
```

1.2.5 Get node name, path and type

```
>>> print node.name
'RenderGraph'
>>> print node.path
'|RenderGraph'
>>> print node.type
'RenderGraph'
```

1.2.6 Get parent node

```
>>> print node.parent
GuerillaNode(44, 'RenderGraph', 'RenderGraph')
```

1.2.7 Get node display name

This is useful for aov nodes.

```
>>> print node.display_name
'Beauty'
```

1.2.8 Iterate over every children of a node

```
>>> for child in node.children:
>>>     print child.path
```

1.2.9 Get node children by its name

```
>>> node.get_child("RenderPass")
```

1.2.10 Get node from its path

```
>>> p.path_to_node('|RenderPass|Layer|Input1')
```

1.2.11 Get node plug from its name

```
>>> node.get_plug('NodePos')
```

1.2.12 Get plug from it's path

```
>>> p.path_to_plug('|RenderPass.BrdfSamples')
```

1.2.13 Iterate over render passes, render ayers and AOVs

```
>>> rp_iter = (n for n in p.nodes if n.type == 'RenderPass')
>>> for rp in rp_iter:
>>>     rl_iter = (n for n in rp.children if n.type == 'RenderLayer')
>>>     for rl in rl_iter:
>>>         aov_iter = (n for n in rp.children if n.type == 'LayerOut')
>>>         for aov in aov_iter:
>>>             print aov.path, aov.display_name
```

1.2.14 Iterate over every plug of a node

```
>>> for plug in node.plugs:
>>>     print plug.path, plug.type
>>>     if plug.input:  # does node plug have incoming plug?
>>>         print plug.input.path, "->", plug.path
>>>     else:  # no incoming plug? get it's value
>>>         print plug.value
>>>     # if this plug is connected to other plug, we print it
>>>     for out_plug in plug.outputs:
>>>         print plug.path, "->", out_plug.path
```

1.3 Reference

`guerilla_parser` module rely on three classes.

1.3.1 Guerilla Parser class

class guerilla_parser.**GuerillaParser**(*content*, *diagnose=False*)

Bases: object

Guerilla .gproject file parser.

Variables

- **objs** (*dict[int, GuerillaNode|GuerillaPlug]*) – Guerilla "object" per id (parsed in `oid[<id>]`).
- **diagnose** (*bool*) – Diagnose mode.

__init__(*content*, *diagnose=False*)

Init the parser.

Parameters

- **content** (*str*) – Raw Guerilla file content to parse.
- **diagnose** (*bool*) – Will print some diagnostic information if True.

classmethod from_file(*path*, **args*, ***kwords*)

Construct parser reading given file *path* content.

This is the main method to use if you want to use the parser.

Parameters **path** (*str*) – Path of the Guerilla file to parse.

Returns Parser filled with content of given *path*.

Return type *GuerillaParser*

has_changed

Return if current parsed file has changed.

A parsed file can be changed using *set_plug_value()* method.

Returns True if both parser instance have same modified content.

Return type bool

modified_content

Modified parsed Guerilla file content.

A parsed file can be changed using *set_plug_value()* method.

Returns Modified parsed Guerilla file content.

Return type str

original_content

Original (unmodified) parsed Guerilla file content.

Returns Original (unmodified) parsed Guerilla file content.

Return type str

write(*path*)

Write modified content to given file *path*.

Parameters **path** (*str*) – File path to write modified content in.

root
Root node (top node of the parsed file).

On standard .gproject files, root node is the *Document*.

Returns Root node.

Return type *GuerillaNode*

doc_format_rev
Document format revision.

Returns Document format revision.

Return type int

Raises **AttributeError** – If no document format revision is present in file.

nodes
Recursively iterate over nodes of the gproject file (except root node).

Returns Generator of nodes of the parsed Guerilla file.

Return type collections.iterator[*GuerillaNode*]

plugs
Iterate over plugs of the gproject file.

Returns Generator of plugs of the parsed Guerilla file.

Return type collections.iterator[*GuerillaPlug*]

path_to_node (*path*)
Find and return node at given *path*.

Example

```
>>> p.path_to_node('|foo|bar|bee')
GuerillaNode('bee', 10, 'primitive')
>>> p.path_to_node('$65|bar|bee')
GuerillaNode('bee', 10, 'primitive')
```

Parameters **path** (*str*) – Path to get node from.

Returns Node found from given *path*.

Return type *GuerillaNode*

Raises

- **PathError** – If root node can't be found.
- **PathError** – If path contain unreachable nodes.

path_to_plug (*path*)
Find and return plug at given *path*.

Example

```
>>> p.path_to_plug('|foo|bar.DiffuseColor')
GuerillaPlug('DiffuseColor', 'Plug', '|foo|bar|bee')
```

Parameters **path** (*str*) – Path to get plug from.

Returns Plug found from given *path*.

Return type *GuerillaPlug*

Raises **PathError** – If path doen't point to a plugs.

static node_to_id_path(*node*)

Return the shortest *id path* for given *node*.

Node id paths are paths relative to the first parent node with an id.

This methode is mostly for internal use to find plug value in `set_plug_value()` but is exposed to the user for his own convinience; debugging, file compare, etc.

Example

```
>>> p.node_to_id_path(my_node)
'$60'
>>> p.node_to_id_path(my_other_node)
'$37|Frustum'
```

In the second line above, `my_other_node` is an implicit node. See *file format information* page for more information.

Parameters **node** (GuerillaNode) – Implicit node to get *id path* from.

Returns

Return type str

set_plug_value(*plug_values*)

While exposed, this method is not stable yet and could potentially change in the future.

Parameters **plug_values** (*list[(GuerillaPlug, str)]*) –

1.3.2 Guerilla Node class

class guerilla_parser.**GuerillaNode**(*id_*, *name*, *type_*, *parent=None*)

Bases: object

Class representing a parsed Guerilla node.

Variables

- **id** (*int*) – Node id (value in parsed expression `oid[<id>]=`).
- **type** (*str*) – Node type.
- **parent** (GuerillaNode) – Node parent.
- **children** (*list[GuerillaNode]*) – Node children.
- **plug_dict** (*dict[str, GuerillaPlug]*) – Node plug by name.

name

Node name.

Returns Node name.

Return type (str, int)

path

Full node path.

Returns Full node path.

Return type str

Raises **PathError** – When node is root.

display_name
Node name shown in UI.

Some nodes (render graph plugs, AOVs, etc.) have a distinction between internal name and UI display name. This property return UI name (aka PlugName attribute) if available.

Returns Node name shown in UI.

Return type str

plugs
Iterator over node plugs.

Returns Iterator over node plugs.

Return type collection.iterator[*GuerillaPlug*]

get_child(*name*)
Return child node with given *name*.

Parameters **name** – Name of the child node to return.

Returns Child node with given *name*.

Return type *GuerillaNode*

Raises **KeyError** – When no child node with given *name* is found.

get_plug(*name*)
Return plug with given *name*.

Parameters **name** – Name of the plug to return.

Returns Plug with given *name*.

Return type *GuerillaPlug*

Raises **KeyError** – When no plug with given name is found

1.3.3 Guerilla Plug class

class guerilla_parser.**GuerillaPlug**(*name*, *type_*, *parent*, *value=None*, *flag=None*, *org_value=None*)

Bases: object

Class representing a parsed Guerilla plug.

Variables

- ***name*** (*str*) – Plug name.
- **type** (*str*) – Plug type (often 'Plug').
- **parent** (GuerillaNode) – Parent plug's node.
- **value** (*bool* | *float* | *str*) – Plug value.
- **org_value** (*str*) – Original parser plug value.
- **input** (GuerillaPlug) – Plug input.
- **outputs** (*list[GuerillaPlug]*) – Plug outputs.

path
Full plug path.

Returns Full plug path.

Return type str

1.4 Known limitations

Guerilla file format imply some parsing limitations.

1.4.1 int vs float

Lua version used in Guerilla (5.1) does not *support* `int` types. Any numeric value is a `float` so it's the attribute that *defines* the type. This also mean a value of `2.0` will always be stored as a `2` in Guerilla files, without the comma.

When Guerilla parse the file back, it store value depending on the attribute type.

This means you can't guess the type from the parsed file only. That's why the parser returns numerical values as Python `float`.

1.4.2 TODO

- object id and oid
- '{}'
- 'create.Id'
- float to int so we only take float
- add delete command support (optional?).
- improve and expose set_plug_value().
- utest to execute inside Guerilla (Guerilla Docker image?).

1.4.3 Conversions

- Missing lua to python conversion *'{}'*.
- Missing lua to python conversion *'types.color'*.
- Missing lua to python conversion *'types.float {min=1,max=10}'*.
- Missing lua to python conversion *'matrix.create{-1,0,0,0,0,1,0,0,0,0,-1,0,0,0,0,1}'*.
- Missing lua to python conversion *'transform.create{-1,0,0,0,0,1,0,0,0,0,-1,0,0,0,0,1}'*.
- Curve unpacking is not supported.

1.4.4 Glayer connection to document

Some `.glayer/.grendergraph` documents have a connection or depend command to root document:

connect("$1.AspectRatio","$0.ProjectAspectRatio") depend("$17.Out","$0|Preferences.ShutterClose")

Such connections are skipped when those files are parsed because document structure doesn't exists.

1.5 Guerilla file format info

Gathered notes about guerilla file format.

1.5.1 Get every class of type Plug

As we are outside Guerilla when parsing, we have to know the classes inherited of `Plug`. To do this, we execute and get the returned list of class from this code snippet:

```
import inspect
import guerilla

for att in dir(guerilla):
    cl = getattr(guerilla, att)
    if not inspect.isclass(cl):
        continue
    if issubclass(cl, guerilla.Plug):
        print cl.__name__
```

This list of class is then put inside global variable `plug_class_names`.

On Guerilla 1.4.17 it list:

- `BakePlug`
- `DynAttrPlug`
- `ExpressionInput`
- `ExpressionOutput`
- `HostPlug`
- `MeshPlug`
- `Plug`
- `UserPlug`

Introspecting Guerilla project files, I found few other, undocumented plug types:

- `HSetPlug`
- `HVisiblePlug`
- `HMattePlug`
- `SceneGraphNodePropsPlug`
- `SceneGraphNodeRenderPropsPlug`
- `AttributePlug`
- `AttributeShaderPlug`

Those types are *local overrides*. Overrides you apply directly on local hierarchy nodes. Those plugs return `guerilla.Plug` when `type()` is called on them.

1.5.2 What are implicit nodes?

When Guerilla need to do a relation/modification to a node which is inside a reference (alembic file for example), he can not do a simple `set("$20.Plug",true)` as the node having the plug (here `$20`) is not inside the gproject but inside the reference.

That's why, Guerilla has to do `set("$19|Foo|Bar.Plug",true)`.

When we parse the file, we have no direct way to know what `Foo` and `Bar` are so we have a special node type `UNKNOWN` to still have nodes without breaking the hierarchy.

Inside the parser code, those are called *implicit nodes*.

Python Module Index

g

guerilla_parser, 9

Index

Symbols

__init__() (*guerilla_parser.GuerillaParser method*), 9

D

display_name (*guerilla_parser.GuerillaNode attribute*), 12
doc_format_rev (*guerilla_parser.GuerillaParser attribute*), 10

F

from_file() (*guerilla_parser.GuerillaParser class method*), 9

G

get_child() (*guerilla_parser.GuerillaNode method*), 12
get_plug() (*guerilla_parser.GuerillaNode method*), 12
guerilla_parser (*module*), 6, 9
GuerillaNode (*class in guerilla_parser*), 11
GuerillaParser (*class in guerilla_parser*), 9
GuerillaPlug (*class in guerilla_parser*), 12

H

has_changed (*guerilla_parser.GuerillaParser attribute*), 9

M

modified_content (*guerilla_parser.GuerillaParser attribute*), 9

N

name (*guerilla_parser.GuerillaNode attribute*), 11
node_to_id_path() (*guerilla_parser.GuerillaParser static method*), 11
nodes (*guerilla_parser.GuerillaParser attribute*), 10

O

original_content (*guerilla_parser.GuerillaParser attribute*), 9

P

path (*guerilla_parser.GuerillaNode attribute*), 11
path (*guerilla_parser.GuerillaPlug attribute*), 12
path_to_node() (*guerilla_parser.GuerillaParser method*), 10
path_to_plug() (*guerilla_parser.GuerillaParser method*), 10
plugs (*guerilla_parser.GuerillaNode attribute*), 12
plugs (*guerilla_parser.GuerillaParser attribute*), 10

R

root (*guerilla_parser.GuerillaParser attribute*), 10

S

set_plug_value() (*guerilla_parser.GuerillaParser method*), 11

W

write() (*guerilla_parser.GuerillaParser method*), 9