pyglreg Documentation

Release 0.9.0a3

Paul Tan

January 14, 2015

Contents

CHAPTER 1

User Guide

1.1 Loading a Registry

Begin by importing the glreg module:

```
>>> import glreg
```

Use `glreg.load()` to load a OpenGL XML API Registry file. Assuming our file is named `gl.xml` in the current directory:

```
>>> registry = glreg.load(open('gl.xml'))
```

`glreg.load()` returns a `glreg.Registry` object.

1.2 Types

`glreg.Type` objects define the OpenGL types such as `GLbyte`, `GLint` etc.

`Registry.types` is a `collections.OrderedDict` object mapping `(type name, type api)` tuples to `Type` objects:

```
>>> registry.types
OrderedDict([(('stddef', None), Type(...)), ...
```

Use `Registry.get_type()` to look up `Type` objects by their name as it will take into account both Types with an API name specified and Types with no API name specified.

```
>>> registry.get_type('GLbyte')  # Get OpenGL's GLbyte typedef
Type('GLbyte', 'typedef signed char {name};')
>>> registry.get_type('GLbyte', 'gles2')  # Get OpenGLES2's GLbyte typedef
Type('GLbyte', 'typedef khronos_int8_t {name};', ...
>>> registry.get_type('GLbyte') is registry.get_type('GLbyte', 'gles2')
False
>>> registry.get_type('GLsync', 'gles2')
Type('GLsync', 'typedef struct __GLsync ...
>>> registry.get_type('GLsync')
Type('GLsync', 'typedef struct __GLsync ...
>>> registry.get_type('GLsync') is registry.get_type('GLsync', 'gles2')
True
```

`Type.template` is the template string of the type in Python's Format String Syntax (**PEP 3101**). It has a *name* replacement field where the type's identifier needs to be substituted in. It is usually `Type.name` or some similar variant.

```
>>> t = registry.get_type('GLbyte')
>>> t.template
```

```
'typedef signed char {name};'
>>> t.template.format(name=t.name)
'typedef signed char GLbyte;'
```

The `Type.text` convenience attribute does this common substitution.

```
>>> t.text
'typedef signed char GLbyte;'
```

Note that `Type` objects can depend on other types. Their names are listed in `Type.required_types`

```
>>> t = registry.get_type('GLbyte', 'gles2')
>>> t.required_types
{'khrplatform'}
```

1.3 Enums

`glreg.Enum` objects define the OpenGL constants such as `GL_POINTS`, `GL_TRIANGLES` etc.

`Registry.enums` is a `collections.OrderedDict` object mapping enum names to `Enum` objects:

```
>>> registry.enums
OrderedDict([('GL_CURRENT_BIT', Enum('GL_CURRENT_BIT', '0x00000001')), ...
>>> registry.enums['GL_POINTS']
Enum('GL_POINTS', '0x0000')
```

1.4 Commands

`glreg.Command` objects define OpenGL functions such as `glClear` and `glDrawArrays`.

`Registry.commands` is a `collections.OrderedDict` object mapping command names to `Command` objects:

```
>>> registry.commands
OrderedDict([('glAccum', Command(...)), ('glAccumxOES', Command(...
>>> registry.commands['glDrawArrays']
Command('glDrawArrays', 'void {name}', [Param('mode', 'GLenum', ...
```

`Command` objects contain their *prototype template* and a list of its parameters as `Param` objects:

```
>>> cmd = registry.commands['glDrawArrays']
>>> cmd.proto_template  # The command's prototype template
'void {name}'
>>> cmd.proto_text  # Convenience attribute for command's prototype
'void glDrawArrays'
>>> cmd.params  # The command's parameters
[Param('mode', 'GLenum', '{type} {name}'), Param('first', 'GLint', ...
```

1.5 Features

`glreg.Feature` objects are basically OpenGL version definitions.

`Registry.features` is a `collections.OrderedDict` object mapping feature names to `Feature` objects.

```
>>> registry.features
OrderedDict([('GL_VERSION_1_0', Feature(...)), ('GL_VERSION_1_1', Feature(...
```

Each `Feature` object lists the type, enum and command names that were introduced in that version in internal `Require` objects.

```
>>> registry.features['GL_VERSION_3_2']  # OpenGL version 3.2
Feature('GL_VERSION_3_2', 'gl', (3, 2), [Require([], ['GL_CONTEXT_CORE_PRO...
>>> feature = registry.features['GL_VERSION_3_2']
>>> feature.requires  # List of Require objects
[Require([], ['GL_CONTEXT_CORE_PROFILE_BIT', 'GL_CONTEXT_COMPATIBILITY...
```

On the other hand, `Remove` objects specify the types, enum and command names that were removed in that version.

```
>>> feature.removes  # List of Remove objects
[Remove([], [], ['glNewList', 'glEndList', 'glCallList', 'glCallLists', ...
```

1.6 Extensions

`glreg.Extension` objects are OpenGL extension definitions. Just like `Feature` objects, each `Extension` object list the type, enum and command names that were defined in that extension in internal `Require` objects.

```
>>> registry.extensions
OrderedDict([('GL_3DFX_multisample', Extension(...)), ('GL_3DFX_tbuffer', ...
```

1.7 Handling dependencies and removals

As seen above, `Feature` objects and `Extension` objects express dependency and removals of types, enums and commands in a registry through their `Require` and `Remove` objects. These dependencies and removals can be resolved using the Registry Importing functions.

`glreg.import_type()` imports a `Type` and its dependencies from one `Registry` object to another.

```
>>> dst_reg = glreg.Registry()
>>> glreg.import_type(dst_reg, registry, 'GLbyte')
>>> dst_reg.types
OrderedDict([(('GLbyte', None), Type('GLbyte', 'typedef signed char ...
>>> dst_reg = glreg.Registry()
>>> glreg.import_type(dst_reg, registry, 'GLbyte', api='gles2')
>>> dst_reg.types
OrderedDict([(('khrplatform', None), Type('khrplatform', ...
```

`glreg.import_enum()` imports a `Enum` from one `Registry` object to another. Note that `Enum` objects have no dependencies.

```
>>> dst_reg = glreg.Registry()
>>> glreg.import_enum(dst_reg, registry, 'GL_POINTS')
>>> dst_reg.enums
OrderedDict([('GL_POINTS', Enum('GL_POINTS', '0x0000'))])
```

`glreg.import_command()` imports a `Command` and its dependencies from one `Registry` to another.

```
>>> dst_reg = glreg.Registry()
>>> glreg.import_command(dst_reg, registry, 'glBufferData')
>>> dst_reg.commands
OrderedDict([('glBufferData', Command('glBufferData', 'vo...
```

`glreg.import_feature()` imports a `Feature` and its dependencies from one `Registry` to another. Removals which are active in the source Registry will be taken into account – all their specified types, enums and commands will not be imported.

```
>>> dst_reg = Registry()
>>> glreg.import_feature(dst, registry, 'GL_VERSION_3_2')
>>> dst_reg.features  # 'dst_reg' now only contains GL_VERSION_3_2 and its deps
OrderedDict([('GL_VERSION_3_2', Feature('GL_VERSION_3_2', 'gl', (3, 2), ...
```

`glreg.import_extension()` imports a `Extension` and its dependencies from one `Registry` to another.

```
>>> dst_reg = Registry()
>>> glreg.import_extension(dst_reg, registry, 'GL_ARB_ES2_compatibility')
>>> dst_reg.extensions
OrderedDict([('GL_ARB_ES2_compatibility', Extension('GL_ARB_ES2_c...
```

1.8 Filtering Features and Extensions

When calling `glreg.import_feature()` without any of its filter arguments, close inspection of the destination registry will reveal that both OpenGL and OpenGL ES commands are mixed together, and that the OpenGL types have overridden the OpenGL ES types. This is undesirable for applications which only target OpenGL and OpenGL ES.

We can ensure that only OpenGL or OpenGL ES types, enums and commands are imported into the destination registry using filters.

`Feature` objects can be filtered by *api name* and *profile name*. `Extension` objects can be filtered by *extension support strings*.

```
>>> dst = Registry()  # Destination registry
>>> import_registry(dst, registry, api='gl', profile='core', support='glcore')
>>> list(dst.features.keys())  # dst now only contains OpenGL Core features
['GL_VERSION_1_0', 'GL_VERSION_1_1', 'GL_VERSION_1_2', ...
>>> list(dst.extensions.keys())  # dst now only contains OpenGL Core extensions
['GL_ARB_ES2_compatibility', 'GL_ARB_ES3_1_compatibility', 'GL_ARB_ES3_comp...
```

`Registry.get_apis()`, `Registry.get_profiles()` and `Registry.get_supports()` will return all the api names, profile names and extension support strings referenced in the registry respectively.

```
>>> sorted(registry.get_apis())
['gl', 'gles1', 'gles2']
>>> sorted(registry.get_profiles())
['common', 'compatibility', 'core']
>>> sorted(registry.get_supports())
['gl', 'glcore', 'gles1', 'gles2']
```

1.9 Grouping Types, Enums and Commands by their Feature or Extension

OpenGL C header files typically group types, enums and commands by the feature or extension where they were first introduced. This can be accomplished using `glreg.group_apis()`.

`glreg.group_apis()` generates a new `Registry` object for every feature and extension in a registry while importing their types, enums and commands. This effectively groups types, enums and commands with the feature or extension where they were first defined.

```
>>> group_apis(registry, api='gles2', support='gles2')
[Registry('GL_ES_VERSION_2_0', OrderedDict([(('khrplatform', None), Type...
```

A simple OpenGL (ES) C header can thus be generated with the following loop:

```
>>> for api in group_apis(registry, api='gles2', support='gles2'):
...     print('#ifndef ' + api.name)
...     print('#define ' + api.name)
...     print(api.text)
...     print('#endif')
#ifndef GL_ES_VERSION_2_0
#define GL_ES_VERSION_2_0
#include <KHR/khrplatform.h>
typedef khronos_int8_t GLbyte;
...
```

1.10 Command-line interface

When run as a script from the command line, glreg provides a simple command line interface for generating C header files from a registry.

Example usage:

```
$ python -mglreg --list-apis gl.xml
gl
gles1
gles2
$ python -mglreg --list-profiles gl.xml
common
compatibility
core
$ python -mglreg --list-supports gl.xml
gl
glcore
gles1
gles2
$ python -mglreg --api gl --profile core --support glcore gl.xml
#ifndef GL_VERSION_1_0
#define GL_VERSION_1_0
typedef void GLvoid;
typedef unsigned int GLenum;
typedef int GLint;
typedef int GLsizei;
typedef double GLdouble;
typedef unsigned int GLbitfield;
typedef float GLfloat;
typedef unsigned char GLboolean;
typedef unsigned int GLuint;
extern void glBlendFunc(GLenum sfactor, GLenum dfactor);
extern void glClear(GLbitfield mask);...
```

The command-line arguments are as follows:

registry
: Registry path. If this argument is not provided, **glreg** will read the registry from standard input.

-o PATH, **--output** PATH
: Write output to *PATH*.

--api API
: Output only features with API name *API*.

--profile PROFILE
: Output only features with profile name *PROFILE*.

--support SUPPORT
: Output only extensions with extension support string *SUPPORT*.

--list-apis
List api names in registry.

--list-profiles
List profile names in registry.

--list-supports
List extension support strings in registry

1.11 Limitations

- `<remove>` tags in `<extension>` tags, despite being defined in the schema, is not supported because they do not make sense.
- `<group>` tags are not supported yet.

API Reference

`glreg` provides functionality to parse and extract data from OpenGL XML API Registry files. Types, enums and functions (commands) in the registry can be enumerated. This module also provides functions to resolve dependencies and filter APIs in the registry. This makes it useful for generating OpenGL headers or loaders.

2.1 Classes

class `glreg.`**`Registry`**(*name=None*, *types=None*, *enums=None*, *commands=None*, *features=None*, *extensions=None*)

API Registry

`name`
Optional Registry name (or None)

`types`
`collections.OrderedDict` mapping of `(type name, type API)` to `Type` objects.

`enums`
`collections.OrderedDict` mapping of enum names to `Enum` objects.

`commands`
`collections.OrderedDict` mapping of command names to `Command` objects.

`features`
`collections.OrderedDict` mapping of feature names to `Feature` objects.

`extensions`
`collections.OrderedDict` mapping of extension names to `Extension` objects.

`text`
(readonly) Formatted API declarations. Equivalent to the concatenation of *text* attributes of all types, enums and commands in this registry.

`get_apis`()
Returns set of api names referenced in this Registry

Returns set of api name strings

`get_extensions`(*support=None*)
Returns filtered list of extensions in this registry

Parameters **support** – Return only extensions with this extension support string, or None to return all extensions.

Returns list of Extension objects

`get_features`(*api=None*)
Returns filtered list of features in this registry

Parameters **api** (*str*) – Return only features with this api name, or None to return all features.

Returns list of Feature objects

get_profiles()
Returns set of profile names referenced in this Registry

Returns set of profile name strings

get_removes(*api=None*, *profile=None*)
Returns filtered list of Remove objects in this registry

Parameters

- **api** (*str*) – Return Remove objects with this api name or None to return all Remove objects.
- **profile** (*str*) – Return Remove objects with this profile or None to return all Remove objects.

Returns list of Remove objects

get_requires(*api=None*, *profile=None*, *support=None*)
Returns filtered list of Require objects in this registry

Parameters

- **api** (*str*) – Return Require objects with this api name or None to return all Require objects.
- **profile** (*str*) – Return Require objects with this profile or None to return all Require objects.
- **support** (*str*) – Return Require objects with this extension support string or None to return all Require objects.

Returns list of Require objects

get_supports()
Returns set of extension support strings referenced in this Registry

Returns set of extension support strings

get_type(*name*, *api=None*)
Returns Type *name*, with preference for the Type of *api*.

Parameters

- **name** (*str*) – Type name
- **api** (*str*) – api name to prefer, of None to prefer types with no api name

Returns Type object

class glreg.**Type**(*name*, *template*, *required_types=None*, *api=None*, *comment=None*)

name
Type name

template
Type definition template with the following arguments:

•*name*: name of the type (usually `Type.name`)

•*apientry*: calling convention macro (usually the string `APIENTRY`, which is a C macro defined by the system platform headers)

required_types
`set` of `str` specifying the names of types this type depends on.

api
API name which this Type is valid for

comment
Optional comment, or None

text
(readonly) Formatted type definition. Equivalent to `self.template.format(name=self.name, apientry='APIENTRY')`

class `glreg.`**Enum**(*name*, *value*, *comment=None*)

name
Enum name

value
Enum string value

comment
Optional comment, or None

text
(readonly) Formatted enum C definition. Equivalent to `'#define {0.name} {0.value}'.format(self)`

class `glreg.`**Command**(*name*, *type*, *proto_template*, *params*, *comment=None*)

name
Command name

type
Command return type, or None

proto_template
Command identifier template string with the following arguments:

- *name*: Command name (usually `Command.name`)
- *type*: Command return type (usually `Command.type`). This argument is only used when this command has a return type.

params
`list` of command `Params`

comment
Optional comment, or None

required_types
(readonly) `set` of names of types which this Command depends on.

proto_text
(readonly) Formatted Command identifier. Equivalent to `self.proto_template.format(type=self.type, name=self.name)`

text
(readonly) Formatted Command C declaration.

class `glreg.`**Param**(*name*, *type*, *template*)

name
Param name

type
Optional name of Param type, or None

template
Param definition template with the following arguments:

- *name*: Param name (usually `Param.name`)
- *type*: Param type (usually `Param.type`). This argument is only used when this Param has a type.

text
Formatted Param definition. Equivalent to `self.template.format(name=self.name, type=self.type)`

class `glreg.`**Feature**(*name*, *api*, *number*, *requires*, *removes*, *comment=None*)
Feature

name
Feature name

api
API name which this Feature is valid for

number
Feature number as `(major, minor)` tuple.

requires
`list` of Feature `Require` objects.

removes
`list` of Feature `Remove` objects.

comment
Optional comment, or None.

get_apis()
Returns set of api names referenced in this Feature.

Returns set of api names

get_profiles()
Returns set of profile names referenced in this Feature

Returns set of profile names

get_removes(*profile=None*)
Get filtered list of Remove objects in this Feature

Parameters **profile** (*str*) – Return Remove objects with this profile or None to return all Remove objects.

Returns list of Remove objects

get_requires(*profile=None*)
Get filtered list of Require objects in this Feature

Parameters **profile** (*str*) – Return Require objects with this profile or None to return all Require objects.

Returns list of Require objects

class `glreg.`**Extension**(*name*, *supported*, *requires*, *comment=None*)
Extension

name
Extension name

supported
`set` of extension 'supported' strings

requires
`list` of `Require` objects

comment
Optional comment, or None.

get_apis()
Returns set of api names referenced in this Extension

Returns set of api name strings

get_profiles()
Returns set of profile names referenced in this Extension

Returns set of profile name strings

get_requires(*api=None*, *profile=None*)
Return filtered list of Require objects in this Extension

Parameters

- **api** (*str*) – Return Require objects with this api name or None to return all Require objects.
- **profile** (*str*) – Return Require objects with this profile or None to return all Require objects.

Returns list of Require objects

get_supports()
Returns set of extension support strings referenced in Extension

Returns set of extension support strings

class glreg.**Require**(*types*, *enums*, *commands*, *profile=None*, *api=None*, *comment=None*)
A requirement

types
`list` of type names which this Require requires.

enums
`list` of enum names which this Require requires.

commands
`list` of command names which this Require requires.

profile
Profile name which this Require is valid for

api
API name which this Require is valid for

comment
Optional comment, or None.

as_symbols()
Set of symbols required by this Require

Returns set of `(symbol type, symbol name)` tuples

class glreg.**Remove**(*types*, *enums*, *commands*, *profile=None*, *comment=None*)
Removal requirement

types
List of type names of Types to remove.

enums
List of enum names of Enums to remove.

commands
List of command names of Commands to remove.

profile
Profile name which this Remove is valid for.

comment
Optional comment, or None.

as_symbols()
Set of symbols required to be removed by this Remove

Returns set of `(symbol type, symbol name)` tuples

2.2 Registry loading functions

glreg.**load**(*f*)
Loads Registry from file

Parameters **f** (*File-like object*) – File to load

Returns Registry

glreg.**loads**(*s*)
Load registry from string

Parameters **s** (*str or bytes*) – Registry XML contents

Returns Registry

2.3 Registry importing functions

glreg.**import_type**(*dest*, *src*, *name*, *api=None*, *filter_symbol=None*)
Import Type *name* and its dependencies from Registry *src* to Registry *dest*.

Parameters

- **dest** (*Registry*) – Destination Registry
- **src** (*Registry*) – Source Registry
- **name** (*str*) – Name of type to import
- **api** (*str*) – Prefer to import Types with api Name *api*, or None to import Types with no api name.
- **filter_symbol** (Callable with signature `(symbol_type:str, symbol_name:str) -> bool`) – Optional filter callable

glreg.**import_enum**(*dest*, *src*, *name*)
Import Enum *name* from Registry *src* to Registry *dest*.

Parameters

- **dest** (*Registry*) – Destination Registry
- **src** (*Registry*) – Source Registry
- **name** (*str*) – Name of Enum to import

glreg.**import_command**(*dest*, *src*, *name*, *api=None*, *filter_symbol=None*)
Import Command *name* and its dependencies from Registry *src* to Registry *dest*

Parameters

- **dest** (*Registry*) – Destination Registry
- **src** (*Registry*) – Source Registry
- **name** (*str*) – Name of Command to import
- **api** (*str*) – Prefer to import Types with api name *api*, or None to import Types with no api name

- **filter_symbol** (Callable with signature `(symbol_type:str, symbol_name:str) -> bool`) – Optional filter callable

`glreg.`**`import_feature`**(*dest*, *src*, *name*, *api=None*, *profile=None*, *filter_symbol=None*)
Imports Feature *name*, and all its dependencies, from Registry *src* to Registry *dest*.

Parameters

- **dest** (*Registry*) – Destination Registry
- **src** (*Registry*) – Source Registry
- **name** (*str*) – Name of Feature to import
- **api** (*str*) – Prefer to import dependencies with api name *api*, or None to import dependencies with no API name.
- **profile** (*str*) – Import dependencies with profile name *profile*, or None to import all dependencies.
- **filter_symbol** (Callable with signature `(symbol_type:str, symbol_name:str) -> bool`) – Optional symbol filter callable

`glreg.`**`import_extension`**(*dest*, *src*, *name*, *api=None*, *profile=None*, *filter_symbol=None*)
Imports Extension *name*, and all its dependencies.

Parameters

- **dest** (*Registry*) – Destination Registry
- **src** (*Registry*) – Source Registry
- **name** (*str*) – Name of Extension to import
- **api** (*str*) – Prefer to import types with API name *api*, or None to prefer Types with no API name.
- **filter_symbol** (Callable with signature `(symbol_type:str, symbol_name:str) -> bool`) – Optional symbol filter callable

`glreg.`**`import_registry`**(*dest*, *src*, *api=None*, *profile=None*, *support=None*, *filter_symbol=None*)
Imports all features and extensions and all their dependencies.

Parameters

- **dest** (*Registry*) – Destination API
- **src** (*Registry*) – Source Registry
- **api** (*str*) – Only import Features with API name *api*, or None to import all features.
- **profile** (*str*) – Only import Features with profile name *profile*, or None to import all features.
- **support** (*str*) – Only import Extensions with this extension support string, or None to import all extensions.
- **filter_symbol** (Callable with signature `(symbol_type:str, symbol_name:str) -> bool`) – Optional symbol filter callable

2.4 API grouping functions

`glreg.`**`group_apis`**(*reg*, *features=None*, *extensions=None*, *api=None*, *profile=None*, *support=None*)
Groups Types, Enums, Commands with their respective Features, Extensions

Similar to `import_registry()`, but generates a new Registry object for every feature or extension.

Parameters

- **reg** (*Registry*) – Input registry
- **features** (*Iterable of strs*) – Feature names to import, or None to import all.
- **extensions** (*Iterable of strs*) – Extension names to import, or None to import all.
- **profile** (*str*) – Import features which belong in *profile*, or None to import all.
- **api** (*str*) – Import features which belong in *api*, or None to import all.
- **support** (*str*) – Import extensions which belong in this extension support string, or None to import all.

Returns list of `Registry` objects

CHAPTER 3

Release Notes

3.1 0.9.0a3

- Critical fix for the bug `group_apis()` which caused it to not return anything because wrong arguments were passed to `import_extension()` and `import_feature()`.
- New attribute `Type.type` to account for Type's return type, if provided. This is reflected in `Type.required_types`. This change fixes the behavior of the registry import API functions.
- `Command.text` has trailing `;` removed.
- The default list of extensions in `group_apis()` are now sorted in the sort order used in the official OpenGL headers.

3.2 0.9.0a2

- Minor metadata changes for PyPI upload

3.3 0.9.0a1

First release. Hello world!

CHAPTER 4

Indices and tables

- *genindex*
- *search*

g

Symbols

A

C

E

F

G

I

L

N

P

R

S

T

V