Celaria Map Toolkit

Release 0.2.0

Iceflower S

Aug 24, 2019

CONTENTS

CHAPTER

ONE

CMT - CELARIA MAP TOOLKIT

Celaria Map Toolkit can convert different map format from one into another.

Install via pip:

```
pip install cmt
```

About the usage see:

```
cmt –help
```

1.1 About

1.1.1 .cmap support

Version	Encode	Decode	Convert	Downgrade	Upgrade
0	✓	✓	✓		✓
1	✓	✓	✓	✓	

1.1.2 .ecmap support

Version	Encode	Decode	Convert	Downgrade	Upgrade
0	✓	✓	✓		✓
1	✓	✓	✓	✓	

1.2 Web

https://github.com/IceflowRE/cmt

1.3 Credits

- **Developer**
  - **Iceflower S**
    - iceflower@gmx.de
- **Format Definition**
  - https://www.celaria.com/

1.3.1 Third Party

pytest

- Holger Krekel and others
- https://github.com/pytest-dev/pytest
- MIT

Prospector

- landscapeio
- https://github.com/landscapeio/prospector
- GPL-2.0+

Read the Docs Sphinx Theme

- Dave Snider
- https://github.com/rtfd/sphinx_rtd_theme
- MIT

Setuptools

- Jason R Coombs / Setuptools Developers
- https://github.com/pypa/setuptools
- MIT

Sphinx

- the Sphinx team
- https://github.com/sphinx-doc/sphinx
- BSD-2-Clause

sphinx-autodoc-typehints

- Alex Grönholm
- https://github.com/agronholm/sphinx-autodoc-typehints
- MIT

twine

- various authors
- https://github.com/pypa/twine

- Apache-2.0

wheel

- Charlie Denton
- https://github.com/meshy/pythonwheels
- BSD-2-Clause

1.3.2 Disclaimer

This software is not official supported by https://www.celaria.com/.

1.3.3 License

Copyright 2019-present Iceflower S (iceflower@gmx.de)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

User Guide

Installation

System Requirements

Python is required in the version 3.7. or higher, it can be downloaded at https://www.python.org/downloads/.

During the Windows installation you should make sure that the `PATH` / environment variable is set and `pip` is installed.

Under Linux it should be ensured that `pip` is installed, if this is not done with the standard installation.

Installation

```
pip install cmt
```

Using cmt

The program is a terminal program, so it runs from the terminal.

Calling with: .. code-block:: none

cmt

-h, --help
show this help message and exit

-v, --version
show program's version number and exit

convert `file {cmap,ecmap} {0,1} output`
convert file to type, version and output file

Internals

Reference material.

cmt

cmt.a_converter

class cmt.a_converter.**AConverter**
Bases: abc.ABC

abstract static convert_to(*source*, *target*)
Convert to the other map format of same version.

Return type *AMap*

abstract static downgrade(*source*)
Downgrade to the format version below.

Return type *AMap*

abstract static upgrade(*source*)
Upgrade to the format version above.

Return type *AMap*

cmt.a_map

class cmt.a_map.**AMap**(*identifier*, *version*)
Bases: abc.ABC

abstract classmethod decode(*data*, *offset*, *debug=False*)

Return type *AMap*

abstract encode()

Return type bytearray

class cmt.a_map.**MapType**
Bases: enum.Enum

An enumeration.

CMAP = 'celaria_map'

ECMAP = 'celaria_edi'

from_str(*text*) **= <function MapType.from_str>**

cmt.convert

cmt.convert.**convert**(*source*, *version*, *target*)

Raises **ValueError** – something failed

Return type Union[*CMap*, *CMap*, *ECMap*, *ECMap*]

cmt.decode

cmt.decode.**decode**(*file*, *debug=False*)

Raises **ValueError** – something failed

Return type Union[*CMap*, *CMap*, *ECMap*, *ECMap*]

cmt.encode

cmt.encode.**encode**(*source*, *file*)

cmt.static_data

Static and important data which is needed by the program and do not need any third party libraries. (this is important because it is used inside the setup.py.

cmt.static_data.**AUTHOR = 'Iceflower S'**
author

cmt.static_data.**AUTHOR_EMAIL = 'iceflower@gmx.de'**
author email

cmt.static_data.**DESCRIPTION = 'Celeria Map Toolkit, can convert different map formats from**
short description

cmt.static_data.**LONG_NAME = 'Celaria Map Toolkit'**
long name of this program

cmt.static_data.**NAME = 'CMT'**
name of this program

cmt.static_data.**PROJECT_URL = 'https://github.com/IceflowRE/cmt'**
project url

cmt.static_data.**VERSION = '0.3.0.dev1'**
version in PEP440 format

cmt.utils

class cmt.utils.**DebugIterUnpack**(*format_*, *buffer*, *what*)
Bases: object

cmt.utils.**debug_print**(*data*, *what*, *value*, *offset=None*)

cmt.utils.**to_hex**(*data*)

cmt.utils.**unpack_from**(*format_*, *buffer*, *offset*, *what*, *debug*)
Same behaviour as struct.unpack_from.

Parameters

- **format** –
- **buffer** (bytes) –
- **offset** (int) –
- **what** (Tuple[str, ...]) – tuple of message for every unpacked value
- **debug** (bool) – use debug mode

Returns

cmt.blender

cmt.blender.v2_80

cmt.blender.v2_80.import_menu

class cmt.blender.v2_80.import_menu.**ImportCMap**(**args*, ***kwargs*)
Bases: bpy.types.Operator, bpy_extras.io_utils.ImportHelper

Import menu for .cmap or .ecmap file.

bl_idname = 'import_scene.cmap'

bl_label = 'Import Celaria Map (.cmap/.ecmap)'

execute(*context*)

Return type Set[str]

filename_ext = '.cmap;.ecmap'

cmt.blender.v2_80.import_menu.**import_cmap**(*ops*, *context*, *filepath*)

cmt.blender.v2_80.import_menu.**menu_func_import**(*self*, *context*)

cmt.blender.v2_80.import_menu.**register**()

cmt.blender.v2_80.import_menu.**unregister**()

cmt.blender.v2_80.object_panel

class cmt.blender.v2_80.object_panel.**BlockProps**(**args*, ***kwargs*)
Bases: bpy.types.PropertyGroup

class cmt.blender.v2_80.object_panel.**Dummy**(**args*, ***kwargs*)
Bases: bpy.types.PropertyGroup

class cmt.blender.v2_80.object_panel.**ObjectPanel**(**args*, ***kwargs*)
Bases: bpy.types.Panel

Creates a Panel in the scene context of the properties editor

bl_context = 'object'

bl_idname = 'OBJECT_PT_cmt'

bl_label = 'Celaria Object'

bl_region_type = 'WINDOW'

bl_space_type = 'PROPERTIES'

draw(*context*)

draw_header(*context*)

classmethod poll(*context*)

cmt.blender.v2_80.object_panel.**register**(*version=1*)

cmt.blender.v2_80.object_panel.**unregister**()

cmt.blender.v2_80.scene_panel

class cmt.blender.v2_80.scene_panel.**MetadataProps**(**args*, ***kwargs*)
Bases: bpy.types.PropertyGroup

class cmt.blender.v2_80.scene_panel.**ScenePanel**(**args*, ***kwargs*)
Bases: bpy.types.Panel

Creates a Panel in the scene context of the properties editor

bl_context = 'scene'

bl_idname = 'SCENE_PT_layout'

bl_label = 'Celaria Metadata'

bl_region_type = 'WINDOW'

bl_space_type = 'PROPERTIES'

draw(*context*)

draw_header(*context*)

cmt.blender.v2_80.scene_panel.**register**()

cmt.blender.v2_80.scene_panel.**unregister**()

cmt.blender.v2_80.utils

cmt.blender.v2_80.utils.**add_exclusive_to_collection**(*obj*, *coll_name*)

cmt.blender.v2_80.utils.**create_material**(*name*, *color*)
Creates a material with a specific color, if the material name does not already exist. :type name: str :param name: :type color: Tuple[float, float, float, float] :param color: :rtype: bpy.types.Material :return:

cmt.blender.v2_80.utils.**get_collection**(*name*)
If the collection is not existing it will create a new. :type name: str :param name: :rtype: bpy.types.Collection :return:

cmt.blender.v2_80.utils.**to_bl_location**(*location*)

Return type Tuple[float, float, float]

cmt.blender.v2_80.utils.**to_bl_rotation_z**(*rotation_z*)

Return type float

cmt.blender.v2_80.utils.**to_bl_scale**(*scale*)

Return type Tuple[float, float, float]

cmt.blender.v2_80.utils.**to_cmt_position**(*location*)

Return type Tuple[float, float, float]

cmt.blender.v2_80.utils.**to_cmt_rotation_z**(*rotation_z*)

Return type float

cmt.blender.v2_80.utils.**to_cmt_scale**(*scale*)

Return type Tuple[float, float, float]

cmt.blender.v2_80.viewport_add_menu

class cmt.blender.v2_80.viewport_add_menu.**AddNewBlock**(**args*, ***kwargs*)
Bases: bpy.types.Operator

Add a new Block

bl_idname = 'cmt.add_new_block'

bl_label = 'Add a new Celaria Block'

execute(*context*)

Return type Set[str]

class cmt.blender.v2_80.viewport_add_menu.**AddNewDummy**(**args*, ***kwargs*)
Bases: bpy.types.Operator

Add a new Dummy

bl_idname = 'cmt.add_new_dummy'

bl_label = 'Add a new Celaria Dummy'

execute(*context*)

Return type Set[str]

class cmt.blender.v2_80.viewport_add_menu.**AddNewPlayerStart**(**args*, ***kwargs*)
Bases: bpy.types.Operator

Add a new Player Start

bl_idname = 'cmt.add_new_playerstart'

bl_label = 'Add a new Celaria Player Start'

execute(*context*)

Return type Set[str]

class cmt.blender.v2_80.viewport_add_menu.**AddNewSphere**(**args*, ***kwargs*)
Bases: bpy.types.Operator

Add a new Sphere

bl_idname = 'cmt.add_new_sphere'

bl_label = 'Add a new Celaria Sphere'

execute(*context*)

Return type Set[str]

class cmt.blender.v2_80.viewport_add_menu.**ViewportAddMenu**(**args*, ***kwargs*)
Bases: bpy.types.Menu

Create new Celaria Objects

bl_idname = 'VIEW3D_MT_add_cmt'

bl_label = 'Celaria Objects'

draw(*context*)

cmt.blender.v2_80.viewport_add_menu.**menu_add_object**(*self*, *context*)

cmt.blender.v2_80.viewport_add_menu.**register**()

cmt.blender.v2_80.viewport_add_menu.**unregister**()

cmt.blender.v2_80.v1

cmt.blender.v2_80.v1.add_objects

cmt.blender.v2_80.v1.add_objects.**add_block**(*ent*, *time=None*)
Add a block to collection. :return Created object.

Return type bpy.types.Object

cmt.blender.v2_80.v1.add_objects.**add_camera**(*position*, *look_at*, *preview_cam_set*)

Return type bpy.types.Object

cmt.blender.v2_80.v1.add_objects.**add_dummy**(*ent*)

Return type bpy.types.Object

cmt.blender.v2_80.v1.add_objects.**add_player_start**(*ent*)

cmt.blender.v2_80.v1.add_objects.**add_sphere**(*ent*)

Return type bpy.types.Object

cmt.blender.v2_80.v1.add_objects.**add_sun**(*angle*, *rotation_z*)

Return type bpy.types.Object

cmt.blender.v2_80.v1.add_objects.**get_color**(*typ*)
Get the color of a specific block.

Return type Tuple[float, float, float, float]

cmt.blender.v2_80.v1.codec

cmt.blender.v2_80.v1.codec.**decode**(*cmap*)

cmt.cmap

cmt.cmap.a_cmap

class cmt.cmap.a_cmap.**ACMap**(*version*)
Bases: *cmt.a_map.AMap*

abstract classmethod decode(*data*, *offset*, *debug=False*)

Return type *ACMap*

abstract encode()

Return type bytearray

cmt.cmap.a_entity

class cmt.cmap.a_entity.**AEntity**(*type_*, *byte_size*)
Bases: abc.ABC

Variables

- **type** – entity type
- **byte_size** – size in bytes the entity uses

abstract classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *AEntity*

abstract encode()
Includes the entity type.

Return type bytearray

cmt.cmap.v0

cmt.cmap.v0.cmap

class cmt.cmap.v0.cmap.**CMap**
Bases: *cmt.cmap.a_cmap.ACMap*

Celaria .cmap format (version 0)

Datatypes

Abbreviation	Type	Byte size
uByte	unsigned byte	1
uShort	unsigned short	2
uInt	unsigned int	4
sShort	signed short	2
sInt	signed int	4
f32	float	4
f64	double	8

Description format

```
> <datatype> (<number of datatypes in sequence>) // <description>
```

or

```
> [<variable name>] :  <datatype> (<number of datatypes in sequence>) //
<description>
```

Format

```
> uByte (11) // string identifier
> uByte (1) // version

> nameLen : uByte (1) // number of characters in map name
> uByte (nameLen) // map name as String

> uByte (1) - boolean, if the timer will be run in singleplayer

> uByte (1) // unused byte

> times : uByte (1) - number of checkpoint times (including medal time)

> uInt (times) // checkpoint times for platin
> uInt (times) // checkpoint times for gold
> uInt (times) // checkpoint times for silver
> uInt (times) // checkpoint times for bronze

> f32 (1) // sun rotation on Z axis
> f32 (1) // sun height expressed as an angle (between 0 and 90 degrees)

> f64 (1) // preview camera position x
> f64 (1) // preview camera position y
> f64 (1) // preview camera position z
> f64 (1) // preview camera look at position x
> f64 (1) // preview camera look at position y
> f64 (1) // preview camera look at position z

> entityNumber : uInt (1) // number of entities on the map

for entity in entityNumber {
    > entityType : uInt (1) // entityType

    switch(entityType) {
        case 0: // block
            > blockType : uByte (1) // blockType/color
            > uByte (1) // unused byte
            > sInt (1) // position x
```

(continues on next page)

(continued from previous page)

```
            > sInt (1) // position y
            > uInt (1) // position z
            > uInt (1) // scale x
            > uInt (1) // scale y
            > uInt (1) // scale z
            > f32 (1) // rotation on Z axis

            if (blockType == 5){ // checkpoint block
                > uByte (1) // checkpoint Number
            }

        case 1: // sphere
            > sInt (1) // position x
            > sInt (1) // position y
            > sInt (1) // position z

        case 2: // player start
            > uByte (1) // unused byte
            > sInt (1) // position x
            > sInt (1) // position y
            > sInt (1) // position z
            > f32 (1) // rotation on Z axis

        case 128: // dummy id
            > uByte (1) // ID
            > sInt (1) // position x
            > sInt (1) // position y
            > uInt (1) // position z
            > uInt (1) // scale x
            > uInt (1) // scale y
            > uInt (1) // scale z
            > f32 (1) // rotation on Z axis
}
```

classmethod decode(*data*, *offset*, *debug=False*)

Return type CMAP

encode()

Return type bytearray

cmt.cmap.v0.entities

class cmt.cmap.v0.entities.**Block**

Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *Block*

encode()
Includes the entity type.

Return type bytearray

class cmt.cmap.v0.entities.**BlockType**
Bases: enum.Enum

An enumeration.

CHECKPOINT = 5

FINISH = 1

ICE = 4

JUMP = 2

NOTHING = 0

SPEED = 3

class cmt.cmap.v0.entities.**Dummy**
Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *Dummy*

encode()
Includes the entity type.

Return type bytearray

class cmt.cmap.v0.entities.**PlayerStart**
Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *PlayerStart*

encode()
Includes the entity type.

Return type bytearray

class cmt.cmap.v0.entities.**Sphere**
Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *Sphere*

encode()
Includes the entity type.

Return type bytearray

cmt.cmap.v0.medal_time

class cmt.cmap.v0.medal_time.**MedalTime**(*platin=0*, *gold=0*, *silver=0*, *bronze=0*)
Bases: object

cmt.cmap.v0.medal_time.**decode_medal_times**(*data*, *offset*, *debug=False*)
Must start with the length byte.

Parameters

- **data** (bytes) –
- **offset** (int) –
- **debug** (bool) –

Return type List[*MedalTime*]

cmt.cmap.v1

cmt.cmap.v1.cmap

class cmt.cmap.v1.cmap.**CMap**
Bases: *cmt.cmap.a_cmap.ACMap*

Celaria .cmap format (version 1)

Datatypes

Abbreviation	Type	Byte size
uByte	unsigned byte	1
uShort	unsigned short	2
uInt	unsigned int	4
sShort	signed short	2
sInt	signed int	4
f32	float	4
f64	double	8

Description format

```
> <datatype> (<number of datatypes in sequence>) // <description>
```

or

```
> [<variable name>] :  <datatype> (<number of datatypes in sequence>) //
<description>
```

Format

```
> uByte (11) // string identifier
> uByte (1) // version

> nameLen : uByte (1) // number of characters in map name
> uByte (nameLen) // map name as String

> uByte (1) // boolean, previewCam_set

> uByte (1) // number of checkpoint times (including finish line)

> times : uByte (1) // number of checkpoint times (including finish line)

> uInt (times) // checkpoint times for platin
> uInt (times) // checkpoint times for gold
> uInt (times) // checkpoint times for silver
> uInt (times) // checkpoint times for bronze

> f32 (1) // sun rotation on Z axis
> f32 (1) // sun angle to xy plane (between 0 and 90 degrees)

> f64 (1) // preview camera position x
> f64 (1) // preview camera position y
> f64 (1) // preview camera position z
> f64 (1) // preview camera look at position x
> f64 (1) // preview camera look at position y
> f64 (1) // preview camera look at position z

> entityNumber : uInt (1) // number of entities on the map

for entity in entityNumber {
    > entityType : uInt (1) // entityType

    switch(entityType) {
        case 0: // block
            > blockType : uByte (1) // blockType/color
            > sInt (1) // position x
            > sInt (1) // position y
            > uInt (1) // position z
            > uInt (1) // scale x
            > uInt (1) // scale y
            > uInt (1) // scale z
            > f32 (1) // rotation on Z axis

            if (blockType == 5){ // checkpoint block
                > uByte (1) // checkpoint Number
            }

        case 1: // sphere
            > sInt (1) // position x
            > sInt (1) // position y
            > uInt (1) // position z

        case 2: // player start
            > uByte (1) // unknown
            > sInt (1) // position x
            > sInt (1) // position y
```

(continues on next page)

(continued from previous page)

```
            > uInt (1) // position z
            > f32 (1) // rotation on Z axis

        case 128: // dummy id
            > uByte (1) // ID
            > sInt (1) // position x
            > sInt (1) // position y
            > uInt (1) // position z
            > uInt (1) // scale x
            > uInt (1) // scale y
            > uInt (1) // scale z
            > f32 (1) // rotation on Z axis
    }
}
```

classmethod decode(*data*, *offset*, *debug=False*)

Return type CMAP

encode()

Return type bytearray

cmt.cmap.v1.entities

class cmt.cmap.v1.entities.**Block**

Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *Block*

encode()

Includes the entity type.

Return type bytearray

class cmt.cmap.v1.entities.**BlockType**

Bases: enum.Enum

An enumeration.

CHECKPOINT = 5

FINISH = 1

ICE = 4

JUMP = 2

NOTHING = 0

SPEED = 3

class cmt.cmap.v1.entities.**Dummy**
Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *Dummy*

encode()
Includes the entity type.

Return type bytearray

class cmt.cmap.v1.entities.**PlayerStart**
Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

Return type *PlayerStart*

encode()
Includes the entity type.

Return type bytearray

class cmt.cmap.v1.entities.**Sphere**
Bases: *cmt.cmap.a_entity.AEntity*

classmethod decode(*data*, *offset*, *debug=False*)

Parameters

- **data** (bytes) –
- **offset** (int) – without entity type byte
- **debug** (bool) –

encode()
Includes the entity type.

Return type bytearray

cmt.cmap.v1.checkpoint_time

class cmt.cmap.v1.checkpoint_time.**CheckpointTime**(*platin=0*, *gold=0*, *silver=0*, *bronze=0*)
Bases: object

cmt.cmap.v1.checkpoint_time.**decode_checkpoint_times**(*data*, *offset*, *debug=False*)
Must start with the length byte.

Parameters

- **data** (bytes) –
- **offset** (int) –
- **debug** (bool) –

Return type List[*CheckpointTime*]

cmt.converter

cmt.converter.v0

class cmt.converter.v0.**Converter**
Bases: *cmt.a_converter.AConverter*

static convert_to(*source*, *target*)
Convert to the other map format of same version.

Return type Union[*CMap*, *ECMap*]

static downgrade(*source*)
Downgrade to the format version below.

Return type None

static upgrade(*source*)
Upgrade to the format version above.

Return type Union[*CMap*, *ECMap*]

cmt.converter.v1

class cmt.converter.v1.**Converter**
Bases: *cmt.a_converter.AConverter*

static convert_to(*source*, *target*)
Convert to the other map format of same version.

Return type Union[*CMap*, *ECMap*]

static downgrade(*source*)
Downgrade to the format version below.

Return type Union[*CMap*, *ECMap*]

static upgrade(*source*)
Upgrade to the format version above.

Return type Union[Forwardref, Forwardref]

cmt.cs

cmt.cs.main

cmt.cs.main.**main**(*argv=None*)
Entry point into the program. Gets the arguments from the console and proceed them with ArgumentParser. Returns if its success successful 0 else 1.

cmt.ecmap

cmt.ecmap.a_ecmap

class cmt.ecmap.a_ecmap.**AECMap**(*version*)
Bases: *cmt.a_map.AMap*

abstract classmethod decode(*data*, *offset*, *debug=False*)

Return type *AECMap*

abstract encode()

Return type bytearray

cmt.ecmap.v0

cmt.ecmap.v0.ecmap

class cmt.ecmap.v0.ecmap.**ECMap**
Bases: *cmt.ecmap.a_ecmap.AECMap*

Celaria .ecmap format (version 0)

Datatypes

Abbreviation	Type	Byte size
uByte	unsigned byte	1
uShort	unsigned short	2
uInt	unsigned int	4
sShort	signed short	2
sInt	signed int	4
f32	float	4
f64	double	8

Description format

```
> <datatype> (<number of datatypes in sequence>) // <description>
```

or

```
> [<variable name>] :  <datatype> (<number of datatypes in sequence>) //
<description>
```

Difference regarding to the .cmap begins with a '!!'.

Format

```
> uByte (11) // string identifier
> uByte (1) // version

> nameLen : uByte (1) // number of characters in map name
> uByte (nameLen) // map name as String

!! // checkpoint times are missing

> f32 (1) // sun rotation on Z axis
> f32 (1) // sun angle to xy plane (between 0 and 90 degrees)

... same as cmap v0 ...
```

classmethod decode(*data*, *offset*, *debug=False*)

Return type ECMAP

encode()

Return type bytearray

cmt.ecmap.v1

cmt.ecmap.v1.ecmap

class cmt.ecmap.v1.ecmap.**ECMap**

Bases: *cmt.ecmap.a_ecmap.AECMap*

Celaria .ecmap format (version 1)

Datatypes

Abbreviation	Type	Byte size
uByte	unsigned byte	1
uShort	unsigned short	2
uInt	unsigned int	4
sShort	signed short	2
sInt	signed int	4
f32	float	4
f64	double	8

Description format

```
> <datatype> (<number of datatypes in sequence>) // <description>
```

or

```
> [<variable name>] :  <datatype> (<number of datatypes in sequence>) //
<description>
```

Difference regarding to the .cmap begins with a '!!'.

Format

```
> uByte (11) // string identifier
> uByte (1) // version
```

(continues on next page)

(continued from previous page)

```
> nameLen : uByte (1) // number of characters in map name
> uByte (nameLen) // map name as String

!! // checkpoint times are missing

> f32 (1) // sun rotation on Z axis
> f32 (1) // sun angle to xy plane (between 0 and 90 degrees)

... same as cmap v1 ...
```

classmethod decode(*data*, *offset*, *debug=False*)

Return type ECMAP

encode()

Return type bytearray

- genindex
- modindex

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P