CraftProtocol Documentation

Release 0.2.7-SNAPSHOT

Toranktto

Sep 09, 2018

Contents

Source code: CraftProtocol

This module provide high level interface for handling Named Binary Tags (NBT) and Minecraft network packets.

> **Warning:** This documentation is not complete.

CHAPTER 1

Named Binary Tags (NBT)

Creating in-memory NBT tags:

```
from CraftProtocol import NBT

tag = NBT.NBTTagCompound()
tag["example_int_list_key"] = NBT.NBTTagList(NBT.NBTTagByte)
tag["example_int_list_key"].append(NBT.NBTTagByte(0x00))
tag["example_int_list_key"].append(NBT.NBTTagByte(0x0F))
tag["example_string_key"] = NBT.NBTTagString("string")
tag["example_double_key"] = NBT.NBTTagDouble(120.1351)
```

Warning: You can't assign a native python value to NBTTagCompound key. Use class that inherits from NBTBase.

Python Type	NBT Type
NoneType	NBTBase
str	NBTTagString
unicode	NBTTagString
int	NBTTagInt
int	NBTTagShort
int	NBTTagByte
long	NBTTagLong
float	NBTTagFloat
float	NBTTagDouble
dict	NBTTagCompound
list	NBTTagList
list	NBTTagByteArray
list	NBTTagIntArray
list	NBTTagLongArray

Remember that NBT Types with "Array" suffix can store only one NBT Type which is prefixed and NBTTagList can store only one choosed by user (in constructor) NBT Type.

To serialize NBT tag to stream (file-like or socket object):

```
from CraftProtocol.NBT import NBTSerializer

NBTSerializer.write(stream, tag)
```

or deserialize from stream:

```
from CraftProtocol.NBT import NBTSerializer

tag = NBTSerializer.read(stream)
```

Note: NBTSerializer does not support compression. If you want to handle compressed NBT tags, use appropriate module for this, for example `gzip`.

To convert NBT Value to Python Value:

```
nbt_string = tag["example_string_key"]
native_string = nbt_string.get()
```

CHAPTER 2

Packets

Documentation of Minecraft network protocol can be found at http://wiki.vg. Currently implemented versions of this protocol:

- 1.8.x
- 1.10.x
- 1.12.2

Note: In described examples, we use 1.10.x protocol.

Creating packet:

```
from CraftProtocol.Protocol import ProtocolVersion
from CraftProtocol.Protocol import ProtocolState
from CraftProtocol.Protocol.v1_10 import Packet

packet = Packet.Handshaking.HandshakePacket(ProtocolVersion.MC_1_10, "hostname",␣
↪25565, ProtocolState.STATUS)
```

To know arguments for packet constructor, see http://wiki.vg.

If you want to serialize packets, use:

```
from CraftProtocol.Protocol import ProtocolVersion
from CraftProtocol.Protocol import Packet

serializer = CraftProtocol.Protocol.Packet.PacketSerializer(
    CraftProtocol.Protocol.ProtocolVersion.MC_1_10,
    CraftProtocol.Protocol.Packet.PacketSerializer.Mode.CLIENT
)
serializer.write(stream, packet)
```

To deserialize packets from stream (e.g. from socket object):

```
from CraftProtocol.Protocol import ProtocolVersion
from CraftProtocol.Protocol import Packet

serializer = CraftProtocol.Protocol.Packet.PacketSerializer(
    CraftProtocol.Protocol.ProtocolVersion.MC_1_10,
    CraftProtocol.Protocol.Packet.PacketSerializer.Mode.CLIENT
)
packet = serializer.read(stream)
```

Warning: Different protocols may have different packets. See http://wiki.vg for details.

Warning: Some packets is not implemented in CraftProtocol (only in `Play` protocol state). If you want to know list of implemented packets, see source code.

Warning: You must manually change serializer protocol state if necessary. Default state is `Handshaking`. For example, in `Server List Ping` after sending Handshake packet state must be switched to `Status`.

Note: Valid packet must inherits from `CraftProtocol.Protocol.Packet.BasePacket` class.

CHAPTER 3

Chat

If you want to strip colors from chat (`dict`) object (e.g. from `ChatMessageClientPacket` packet) use:

```
from CraftProtocol.Chat import ChatSerializer

raw_text = ChatSerializer.strip_colors(chat)
```

To serialize chat in legacy format use:

```
from CraftProtocol.Chat import ChatSerializer

chat = ChatSerializer.translate_legacy(raw_text)
```

Note: Serializing chat in modern (JSON) format is not currently supported.

CHAPTER 4

Chat Objects

class CraftProtocol.Chat.**ChatMode**

Note: This class is used to enum purposes only. You don't have to create new instances.

ENABLED
This is default mode.

HIDDEN
In this mode, chat is hidden.

COMMANDS
In this mode, only commands can be send to server.

class CraftProtocol.Chat.**ChatSerializer**

Note: This class is static. You don't have to create new instances.

static strip_colors(*chat*)
Strip colors in chat (`dict`) message and return it.

Parameters **chat** (*dict*) – chat message in modern JSON format

static translate_legacy(*text*, *code="&"*)
Translate colored chat message (like this `&aHello &bWorld`) to paragraph-prefixed legacy format.

Parameters

- **text** – text to translate
- **code** – character that is replaced to paragraph

Return type unicode

CHAPTER 5

Inventory Objects

class CraftProtocol.Inventory.**Inventory**(*window_id*, *title*, *inventory_type*, *slots_number*, *entity_id=None*)

Parameters

- **window_id** (*int*) – inventory window id
- **title** (*dict*) – inventory title in chat format
- **inventory_type** (*basestring enum*) – inventory type
- **slots_number** (*int*) – number of items in inventory
- **entity_id** (*int*) – only used if inventory type is EntityHorse.

Note: Currently there is no class that has inventory type enum defined.

get_window_id()
Return inventory window id.

Return type int

get_title()
Return inventory title.

Return type dict

Note: Inventory title uses Chat format.

get_type()
Return inventory type.

Return type unicode enum

Note: Currently there is no class that has this enum defined.

get_slots()
Return items in this inventory.

Return type list (*CraftProtocol.Inventory.SlotData*)

get_entity_id()
Return inventory entity id.

Return type int or None

Note: Used only when Inventory Type is `EntityHorse`.

__getitem__(*index*)
Return item at specified index.

Parameters **index** (*int*) – index

Return type *CraftProtocol.Inventory.SlotData*

__setitem__(*index*, *value*)
Set item at specified index.

Parameters

- **index** (*int*) – index
- **value** (`CraftProtocol.Inventory.SlotData`) – slot data

__delitem__(*index*)
Set item at specified index to empty slot.

Parameters **index** (*int*) – index

__len__()
Return number of items in this inventory.

Return type int

__iter__()
The same as __iter__() of get_slots().

Return type iter

copy()
Return copy of this inventory.

Return type *CraftProtocol.Inventory.Inventory*

class CraftProtocol.Inventory.**SlotData**(*item_id*, *count=0*, *damage=0*, *tag=None*)

Parameters

- **item_id** (*int*) – item id
- **count** (*int*) – item count
- **damage** (*int*) – item variant
- **tag** (*CraftProtocol.NBT.NBTTagCompound or None*) – item NBT tag

static empty()
Return empty slot.

Return type *CraftProtocol.Inventory.SlotData*

get_id()
Return slot item id.

Return type int

get_count()
Return number of items in this slot.

Return type int

set_count(*count*)
Set number of items in this slot.

Parameters **count** (*int*) – items count

get_damage()
Return slot item variant.

Return type int

set_damage(*damage*)
Set slot item variant.

Parameters **damage** (*int*) – item variant

get_tag()
Return NBT tag of this slot.

Return type CraftProtocol.NBT.NBTTagCompound or None

set_tag(*tag*)
Set NBT tag of this slot.

Parameters **tag** (*CraftProtocol.NBT.NBTTagCompound or None*) – NBT tag

has_tag()
Return `True` if slot has NBT tag.

Return type bool

CHAPTER 6

World Objects

class CraftProtocol.World.**World**(*world_type*, *dimension*, *difficulty*)

Parameters

- **world_type** (*CraftProtocol.World.WorldType basestring enum*) – world type
- **dimension** (*CraftProtocol.World.WorldDimension int enum*) – world dimension
- **difficulty** (*CraftProtocol.World.WorldDifficulty int enum*) – world difficulty

get_world_type()
Return world type.

Return type CraftProtocol.World.WorldType unicode enum

get_dimension()
Return world dimension.

Return type CraftProtocol.World.WorldDimension int enum

get_difficulty()
Return world difficulty.

Return type CraftProtocol.World.WorldDifficulty int enum

class CraftProtocol.World.**Location**(*x*, *y*, *z*, *yaw=0.00*, *pitch=0.00*)

Parameters

- **x** (*float*) – x
- **y** (*float*) – y
- **z** (*float*) – z
- **yaw** (*float*) – yaw
- **pitch** (*float*) – pitch

get_x()
Return x.

Return type float

set_x(*x*)
Set x.

Parameters **x** (*float*) – new x

get_y()
Return y.

Return type float

set_y(*y*)
Set y.

Parameters **y** (*float*) – new y

get_z()
Return z.

Return type float

set_z(*z*)
Set z.

Parameters **z** (*float*) – new z

get_yaw()
Return yaw.

Return type float

set_yaw(*yaw*)
Set yaw.

Parameters **yaw** (*float*) – new yaw

get_pitch()
Return pitch.

Return type float

set_pitch(*pitch*)
Set pitch.

Parameters **pitch** – new pitch

class CraftProtocol.World.**WorldDimension**

Note: This class is used to enum purposes only. You don't have to create new instances.

NETHER
Represent Nether dimension.

OVERWORLD
Represent natural dimension.

END
Represent The End dimension.

class CraftProtocol.World.**WorldDifficulty**

Note: This class is used to enum purposes only. You don't have to create new instances.

PEACEFUL
Represent easiest difficulty.

EASY
Represent easy difficulty.

NORMAL
Represent normal difficulty.

HARD
Represent hard difficulty.

class CraftProtocol.World.**WorldType**

Note: This class is used to enum purposes only. You don't have to create new instances.

DEFAULT
Represent default map type.

FLAT
Represent super flat map type.

LARGE_BIOMES
Represent map type with large biomes.

AMPLIFIED
Represent amplified map type.

DEFAULT_1_1
Represent legacy map type.

CHAPTER 7

Protocol Objects

class CraftProtocol.Protocol.**ProtocolState**

Note: This class is used to enum purposes only. You don't have to create new instances.

HANDSHAKING
Represent `Handshaking` protocol state.

STATUS
Represent `Status` (`Server List Ping`) protocol state.

LOGIN
Represent `Login` protocol state.

PLAY
Represent `Play` protocol state.

class CraftProtocol.Protocol.**ProtocolVersion**

Note: This class is used to enum purposes only. You don't have to create new instances.

MC_1_8
Represent protocol version that is used by Minecraft 1.8.x.

MC_1_10
Represent protocol version that is used by Minecraft 1.10.x.

MC_1_12_2
Represent protocol version that is used by Minecraft 1.12.2.

Python Module Index

C

Index

Symbols

A

C

D

E

F

G

H

L

M

N

O

P

S

T