Xbox-Smartglass-Core Documentation

Release 1.3.0

OpenXbox

Jun 11, 2021

CONTENTS:

1 Console - The API to use **1**

2 Managers - Communicate with the Service Channels **7**

3 CoreProtocol - The inner workings **13**

4 Cryptography **21**

5 Packet Factory - Assemble packets **25**

6 xbox.auxiliary package **31**
- 6.1 Subpackages . . . 31
- 6.2 Submodules . . . 31
- 6.3 Module contents . . . 34

7 xbox.stump package **35**
- 7.1 Submodules . . . 35
- 7.2 Module contents . . . 41

8 Indices and tables **43**

Python Module Index **45**

Index **47**

CHAPTER

ONE

CONSOLE - THE API TO USE

Console

Console class is a wrapper around the protocol, can be used to *discover*, *poweron* and *connect*. Also stores various device status informations.

It can be either created manually or instantiated via *DiscoveryResponse* message. However, calling static method *discover* does all that for you automatically.

Example

Discovery and connecting:

```
import sys
from xbox.sg.console import Console
from xbox.sg.enum import ConnectionState

discovered = await Console.discover(timeout=1)
if len(discovered):
    console = discovered[0]
    await console.connect()
    if console.connection_state != ConnectionState.Connected:
        print("Connection failed")
        sys.exit(1)
    await console.wait(1)
else:
    print("No consoles discovered")
    sys.exit(1)

... do stuff ...
```

class xbox.sg.console.**Console**(*address: str, name: str, uuid: uuid.UUID, liveid: str, flags: xbox.sg.enum.PrimaryDeviceFlag = <PrimaryDeviceFlag.Null: 0>, last_error: int = 0, public_key: cryptography.hazmat.primitives.asymmetric.ec.EllipticCurvePublicKey = None*)

Bases: object

__init__(*address: str, name: str, uuid: uuid.UUID, liveid: str, flags: xbox.sg.enum.PrimaryDeviceFlag = <PrimaryDeviceFlag.Null: 0>, last_error: int = 0, public_key: cryptography.hazmat.primitives.asymmetric.ec.EllipticCurvePublicKey = None*)

Initialize an instance of Console

Parameters

- **address** – IP address of console.
- **flags** – Primary device flags
- **name** – Name of console.
- **uuid** – UUID of console.
- **liveid** – Live ID of console.
- **public_key** – Console's Public Key.

async static wait(*seconds: int*) → None
Wrapper around *asyncio.sleep*

Parameters **seconds** – Seconds to wait.

Returns: None

async _ensure_protocol_started() → None
Regular protocol instance, setup with crypto and destination address. Targeted at communication with a specific console.

Returns None

async classmethod _ensure_global_protocol_started() → None
Global protocol instance, used for network wide discovery and poweron.

classmethod from_message(*address: str*, *msg: xbox.sg.utils.struct.XStruct*)
Initialize the class with a *DiscoveryResponse*.

Parameters

- **address** – IP address of the console
- **msg** – Discovery Response struct

Returns: Console instance

classmethod from_dict(*d: dict*)

to_dict() → dict

add_manager(*manager: Type[*xbox.sg.manager.Manager*]*, **args*, ***kwargs*)
Add a manager to the console instance.

This will inherit all public methods of the manager class.

Parameters

- **manager** – Manager to add
- ***args** – Arguments
- ****kwargs** – KwArguments

Returns None

async classmethod discover(**args*, ***kwargs*) → List
Discover consoles on the network.

Parameters

- ***args** –
- ****kwargs** –

Returns List of discovered consoles.

Return type list

classmethod discovered() → List
Get list of already discovered consoles.

Returns List of discovered consoles.

Return type list

async classmethod power_on(*liveid: str*, *addr: Optional[str] = None*, *tries=2*) → None
Power On console with given Live ID.

Optionally the IP address of the console can be supplied, this is useful if the console is stubborn and does not react to broadcast / multicast packets (due to routing issues).

Parameters

- **liveid** (*str*) – Live ID of console.
- **addr** (*str*) – IP address of console.
- **tries** (*int*) – Poweron attempts, default: 2.

Returns: None

async send_message(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel = <ServiceChannel.Core: 0>*, *addr: Optional[str] = None*, *blocking: bool = True*, *timeout: int = 5*, *retries: int = 3*) → Optional[xbox.sg.utils.struct.XStruct]
Send message to console.

Parameters

- **msg** – Unassembled message to send
- **channel** – Channel to send the message on, Enum member of *ServiceChannel*
- **addr** – IP address of target console
- **blocking** – If set and *msg* is *Message*-packet, wait for ack
- **timeout** – Seconds to wait for ack, only useful if *blocking* is *True*
- **retries** – Max retry count.

Returns: None

async json(*data: str*, *channel: xbox.sg.enum.ServiceChannel*) → None
Send json message

Parameters

- **data** – JSON dict
- **channel** – Channel to send the message to

Returns: None

async connect(*userhash: Optional[str] = None*, *xsts_token: Optional[str] = None*) → xbox.sg.enum.ConnectionState
Connect to the console

If the connection fails, error will be stored in *self.connection_state*

Raises **ConnectionException** – If no authentication data is supplied and console disallows anonymous connection.

Returns: Connection state

async launch_title(*uri: str*, *location: xbox.sg.enum.ActiveTitleLocation = <ActiveTitleLocation.Full: 0>*) → xbox.sg.enum.AckStatus

Launch a title by URI

Parameters

- **uri** – Launch uri
- **location** – Target title location

Returns: Ack status

async game_dvr_record(*start_delta: int*, *end_delta: int*) → xbox.sg.enum.AckStatus

Start Game DVR recording

Parameters

- **start_delta** – Start time
- **end_delta** – End time

Returns: Ack status

async disconnect() → None

Disconnect from console.

This will reset connection-, pairing-state, ActiveSurface and ConsoleStatus.

Returns: None

async power_off() → None

Power off the console.

No need to disconnect after.

Returns: None

_handle_message(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*) → None

Internal handler for console specific messages aka. *PairedIdentityStateChange*, *ConsoleStatus* and *ActiveSurfaceChange*.

Parameters

- **msg** – Message data
- **channel** – Service channel

Returns None

_handle_json(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*) → None

Internal handler for JSON messages

Parameters

- **msg** – JSON message instance
- **channel** – Service channel originating from

Returns: None

_handle_timeout() → None

Internal handler for console connection timeout.

Returns: None

async _reset_state() → None
Internal handler to reset the inital state of the console instance.

Returns: None

property public_key
Console's public key.

Returns: Foreign public key

property device_status
Current availability status

Returns DeviceStatus

property connection_state
Current connection state

Returns: Connection state

property pairing_state
Current pairing state

Returns PairedIdentityState

property console_status
Console status aka. kernel version, active titles etc.

Returns XStruct

property active_surface
Currently active surface

Returns XStruct

property available
Check whether console is available aka. discoverable

Returns: *True* if console is available, *False* otherwise

property paired
Check whether client is paired to console

Returns: *True* if console is paired, *False* otherwise

property connected
Check whether client is successfully connected to console

Returns: *True* if connected, *False* otherwise

property authenticated_users_allowed
Check whether authenticated users are allowed to connect

Returns: *True* if authenticated users are allowed, *False* otherwise

property console_users_allowed
Check whether console users are allowed to connect

Returns: *True* if console users are allowed, *False* otherwise

property anonymous_connection_allowed
Check whether anonymous connection is allowed

Returns: *True* if anonymous connection is allowed, *False* otherwise

property is_certificate_pending
Check whether certificate is pending

Returns: *True* if certificate is pending, *False* otherwise

CHAPTER

TWO

MANAGERS - COMMUNICATE WITH THE SERVICE CHANNELS

Managers for handling different ServiceChannels.

If a manager for a specific `ServiceChannel` is attached, incoming messages get forwarded there, otherways they are discarded.

Managers can be attached by calling *add_manager()* on the `Console` object (see example) Methods of manager are available through console-context.

Example

How to add a manager:

```
discovered = await Console.discover(timeout=1)
if len(discovered):
    console = discovered[0]

    # Add manager, optionally passing initialization parameter
    some_arg_for_manager_init = 'example'
    console.add_manager(
        MediaManager,
        additional_arg=some_arg_for_manager_init
    )

    await console.connect()
    if console.connection_state != ConnectionState.Connected:
        print("Connection failed")
        sys.exit(1)
    console.wait(1)

    # Call manager method
    console.media_command(0x54321, MediaControlCommand.PlayPauseToggle, 0)

else:
    print("No consoles discovered")
    sys.exit(1)
```

class xbox.sg.manager.**Manager**(*console*, *channel: xbox.sg.enum.ServiceChannel*)

Bases: `object`

__init__(*console*, *channel: xbox.sg.enum.ServiceChannel*)

Don't use directly! INTERNALLY called by the parent `Console`!

Parameters

- **console** – Console object, internally passed by `Console.add_manager

- **channel** – Service channel

_on_message(*msg*, *channel*)
Managers must implement this

_on_json(*data*, *channel*)
Managers must implement this

async _send_message(*msg: xbox.sg.utils.struct.XStruct*)
Internal method to send messages to initialized Service Channel

Parameters **msg** (XStructObj) – Message

async _send_json(*data: str*) → None
Internal method to send JSON messages to initialized Service Channel

Parameters **data** – JSON message

exception xbox.sg.manager.**InputManagerError**
Bases: Exception

Exception thrown by InputManager

class xbox.sg.manager.**InputManager**(*console*)
Bases: *xbox.sg.manager.Manager*

__init__(*console*)
Input Manager (ServiceChannel.SystemInput)

Parameters **console** – Console object, internally passed by `Console.add_manager

_on_message(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*) → None
Internal handler method to receive messages from SystemInput Channel

Parameters

- **msg** – Message
- **channel** – Service channel

async gamepad_input(*buttons: xbox.sg.enum.GamePadButton*, *l_trigger: int = 0*, *r_trigger: int = 0*, *l_thumb_x: int = 0*, *l_thumb_y: int = 0*, *r_thumb_x: int = 0*, *r_thumb_y: int = 0*) → None
Send gamepad input

Parameters

- **buttons** – Gamepad buttons bits
- **l_trigger** – Left trigger value
- **r_trigger** – Right trigger value
- **l_thumb_x** – Left thumbstick X-axis value
- **l_thumb_y** – Left thumbstick Y-axis value
- **r_thumb_x** – Right thumbstick X-axis value
- **r_thumb_y** – Right thumbstick Y-axis value

Returns: None

exception xbox.sg.manager.**MediaManagerError**
Bases: Exception

Exception thrown by MediaManager

class xbox.sg.manager.**MediaManager**(*console*)

Bases: *xbox.sg.manager.Manager*

__init__(*console*)

Media Manager (ServiceChannel.SystemMedia)

Args: Console object, internally passed by `Console.add_manager

_on_message(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*) → None

Internal handler method to receive messages from SystemMedia Channel

Parameters

- **msg** – Message
- **channel** – Service channel

property media_state

Media state payload

Returns: Media state payload

property active_media

Check whether console has active media

Returns: *True* if media is active, *False* if not

property title_id

Title Id of active media

Returns: Title Id

property aum_id

Application user model Id of active media

Returns: Aum Id

property asset_id

Asset Id of active media

Returns: Asset Id

property media_type

Media type of active media

Returns: Media type

property sound_level

Sound level of active media

Returns: Sound level

property enabled_commands

Enabled MediaCommands bitmask

Returns: Bitmask of enabled commands

property playback_status

Playback status of active media

Returns: Playback status

property rate

Playback rate of active media

Returns: Playback rate

property position
Playback position of active media

Returns: Playback position in microseconds

property media_start
Media start position of active media

Returns: Media start position in microseconds

property media_end
Media end position of active media

Returns: Media end position in microseconds

property min_seek
Minimum seek position of active media

Returns: Minimum position in microseconds

property max_seek
Maximum seek position of active media

Returns: Maximum position in microseconds

property metadata
Media metadata of active media

Returns: Media metadata

async media_command(*title_id: int*, *command: xbox.sg.enum.MediaControlCommand*, *request_id: int = 0*, *seek_position: Optional[int] = None*) → None
Send media command

Parameters

- **title_id** – Title Id
- **command** – Media Command
- **request_id** – Incrementing Request Id
- **seek_position** – Seek position

Returns: None

exception xbox.sg.manager.**TextManagerError**
Bases: Exception

Exception thrown by TextManager

class xbox.sg.manager.**TextManager**(*console*)
Bases: *xbox.sg.manager.Manager*

__init__(*console*)
Text Manager (ServiceChannel.SystemText)

Parameters **console** – Console object, internally passed by `Console.add_manager

_on_message(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*)
Internal handler method to receive messages from SystemText Channel

Parameters

- **msg** (XStructObj) – Message
- **channel** (ServiceChannel) – Service channel

property got_active_session
Check whether a text session is active

Returns Returns *True* if any text session is active, *False* otherwise

Return type bool

property current_text_version
Current Text version

Returns Current Text Version

Return type int

property text_session_id
Current Text session id

Returns Text session id if existing, *None* otherwise

Return type int

property text_options
Current Text options

Returns Text options if existing, *None* otherwise

Return type `TextOption`

property text_input_scope
Current Text input scope

Returns: Text input scope if existing, *None* otherwise

property max_text_length
Maximum Text length

Returns: Max text length if existing, *None* otherwise

property text_locale
Test

Returns: Text locale if existing, *None* otherwise

property text_prompt
Test

Returns: Text prompt if existing, *None* otherwise

reset_session() → None
Delete cached text-session config, -input and -ack messages

Returns: None

async finish_text_input() → None
Finishes current text session.

Returns None

async send_systemtext_input(*text: str*) → Optional[xbox.sg.enum.AckStatus]
Sends text input

Parameters **text** – Text string to send

Raises ***TextManagerError*** – If message was not acknowledged via AckMsg or SystemTextAck

Returns: Ack status

async send_systemtext_ack(*session_id: int*, *version: int*) → Optional[xbox.sg.enum.AckStatus]

Acknowledges a SystemText message sent from the console

Parameters

- **session_id** – Current text session id
- **version** – Text version to ack

Returns: Ack status

async send_systemtext_done(*session_id: int*, *version: int*, *flags: int*, *result: xbox.sg.enum.TextResult*) → Optional[xbox.sg.enum.AckStatus]

Informs the console that a text session is done.

Result field tells wether text input should be accepted or cancelled.

Parameters

- **session_id** – Current text session id
- **version** – Last acknowledged text version
- **flags** – Flags
- **result** – Text result to send

Returns: Ack status

CHAPTER

THREE

COREPROTOCOL - THE INNER WORKINGS

Smartglass protocol core

NOTE: Should not be used directly, use *Console* !

exception xbox.sg.protocol.**ProtocolError**
Bases: Exception

Exception thrown by CoreProtocol

class xbox.sg.protocol.**SmartglassProtocol**(*address: Optional[str] = None*, *crypto_instance: Optional[xbox.sg.crypto.Crypto] = None*)
Bases: asyncio.protocols.DatagramProtocol

HEARTBEAT_INTERVAL = 3.0

__init__(*address: Optional[str] = None*, *crypto_instance: Optional[xbox.sg.crypto.Crypto] = None*)
Instantiate Smartglass Protocol handler.

Parameters

- **address** – Address
- **crypto_instance** – Crypto instance

async stop() → None
Dummy

connection_made(*transport: asyncio.transports.DatagramTransport*) → None
Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data_received() calls. When the connection is closed, connection_lost() is called.

error_received(*exc: OSError*)
Called when a send or receive operation raises an OSError.

(Other than BlockingIOError or InterruptedError.)

connection_lost(*exc: Optional[Exception]*)
Called when the connection is lost or closed.

The argument is an exception object or None (the latter meaning a regular EOF is received or the connection was aborted or closed).

async send_message(*msg*, *channel=<ServiceChannel.Core: 0>*, *addr: Optional[str] = None*, *blocking: bool = True*, *timeout: int = 5*, *retries: int = 3*) → Optional[xbox.sg.utils.struct.XStruct]
Send message to console.

Packing and encryption happens here.

Parameters

- **msg** – Unassembled message to send
- **channel** – Channel to send the message on, Enum member of *ServiceChannel*
- **addr** – IP address of target console
- **blocking** – If set and *msg* is *Message*-packet, wait for ack
- **timeout** – Seconds to wait for ack, only useful if *blocking* is *True*
- **retries** – Max retry count.

Returns: None

Raises **ProtocolError** – On failure

async _send(*data: bytes*, *target: Tuple[str, int]*)
Send data on the connected transport.

If addr is not provided, the target address that was used at the time of instantiating the protocol is used. (e.g. asyncio.create_datagram_endpoint in Console-class).

Parameters

- **data** – Data to send
- **target** – Tuple of (ip_address, port)

datagram_received(*data: bytes*, *addr: str*) → None
Handle incoming smartglass packets

Parameters

- **data** – Raw packet
- **addr** – IP address of sender

Returns: None

async _await_ack(*identifier: str*, *timeout: int = 5*) → Optional[xbox.sg.utils.struct.XStruct]
Wait for acknowledgement of message

Parameters

- **identifier** – Identifier of ack
- **timeout** – Timeout in seconds

Returns Event

Return type Event

_set_result(*identifier: str*, *result: Union[xbox.sg.enum.AckStatus, xbox.sg.utils.struct.XStruct]*) → None
Called when an acknowledgement comes in, unblocks *_await_ack*

Parameters

- **identifier** – Identifier of ack
- **result** – Ack status

Returns: None

async _heartbeat_task() → None
Task checking for console activity, firing *on_timeout*-event on timeout.

Heartbeats are empty "ack" messages that are to be ack'd by the console

Returns None

_on_message(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*) → None
Handle msg of type *Message*.

Parameters

- **msg** – Message
- **channel** – Channel the message was received on

Returns: None

_on_ack(*msg: xbox.sg.utils.struct.XStruct*) → None
Process acknowledgement message.

Parameters **msg** – Message

Returns: None

_on_json(*msg: xbox.sg.utils.struct.XStruct*, *channel: xbox.sg.enum.ServiceChannel*) → None
Process json message.

Parameters

- **msg** – Message
- **channel** – Channel the message was received on

Returns: None

async discover(*addr: str = None*, *tries: int = 5*, *blocking: bool = True*, *timeout: int = 5*) → Dict[str, xbox.sg.utils.struct.XStruct]
Discover consoles on the network

Parameters

- **addr** (*str*) – IP address
- **tries** (*int*) – Discover attempts
- **blocking** (*bool*) – Wait a given time for responses, otherwise return immediately
- **timeout** (*int*) – Timeout in seconds (only if *blocking* is *True*)

Returns List of discovered consoles

Return type list

property discovered
Return discovered consoles

Returns Discovered consoles

async connect(*userhash: str*, *xsts_token: str*, *client_uuid: uuid.UUID = UUID('cf21c4a1-99cd-47c9-9fd4-c38626d33ca4')*, *request_num: int = 0*, *retries: int = 3*) → xbox.sg.enum.PairedIdentityState
Connect to console

Parameters

- **userhash** – Userhash from Xbox Live Authentication
- **xsts_token** – XSTS Token from Xbox Live Authentication

- **client_uuid** – Client UUID (default: Generate random uuid)
- **request_num** – Request number
- **retries** – Max. connect attempts

Returns: Pairing State

Raises ***ProtocolError*** – If connection fails

async local_join(*client_info: Union[xbox.sg.constants.WindowsClientInfo, xbox.sg.constants.AndroidClientInfo] = <class 'xbox.sg.constants.WindowsClientInfo'>*, ***kwargs*) → None

Pair client with console.

Parameters

- **client_info** – Either *WindowsClientInfo* or *AndroidClientInfo*
- ****kwargs** –

Returns: None

async start_channel(*channel: xbox.sg.enum.ServiceChannel*, *messagetarget_uuid: uuid.UUID*, *title_id: int = 0*, *activity_id: int = 0*, ***kwargs*) → None

Request opening of specific ServiceChannel

Parameters

- **channel** – Channel to start
- **messagetarget_uuid** – Message Target UUID
- **title_id** – Title ID, Only used for ServiceChannel.Title
- **activity_id** – Activity ID, unknown use-case
- ****kwargs** – KwArgs

Returns: None

async ack(*processed: List[int]*, *rejected: List[int]*, *channel: xbox.sg.enum.ServiceChannel*, *need_ack: bool = False*) → None

Acknowledge received messages that have *need_ack* flag set.

Parameters

- **processed** – Processed sequence numbers
- **rejected** – Rejected sequence numbers
- **channel** – Channel to send the ack on
- **need_ack** – Whether we want this ack to be acknowledged by the target participant. Will be blocking if set. Required for heartbeat messages.

Returns: None

async json(*data: str*, *channel: xbox.sg.enum.ServiceChannel*) → None

Send json message

Parameters

- **data** – JSON dict
- **channel** – Channel to send the message to

Returns: None

async power_on(*liveid: str*, *addr: Optional[str] = None*, *tries: int = 2*) → None
Power on console.

Parameters

- **liveid** – Live ID of console
- **addr** – IP address of console
- **tries** – PowerOn attempts

Returns: None

async power_off(*liveid: str*) → None
Power off console

Parameters **liveid** – Live ID of console

Returns: None

async disconnect(*reason: xbox.sg.enum.DisconnectReason = <DisconnectReason.Unspecified: 0>*, *error: int = 0*) → None
Disconnect console session

Parameters

- **reason** – Disconnect reason
- **error** – Error Code

Returns: None

async game_dvr_record(*start_delta: int*, *end_delta: int*) → xbox.sg.enum.AckStatus
Start Game DVR recording

Parameters

- **start_delta** – Start time
- **end_delta** – End time

Returns: Acknowledgement status

async launch_title(*uri: str*, *location: xbox.sg.enum.ActiveTitleLocation = <ActiveTitleLocation.Full: 0>*) → xbox.sg.enum.AckStatus
Launch title via URI

Parameters

- **uri** – Uri string
- **location** – Location

Returns: Ack status

class xbox.sg.protocol.**SequenceManager**
Bases: object

__init__()
Process received messages by sequence numbers. Also add processed / rejected messages to a list. Tracks the *Low Watermark* that's sent with *Acknowledgement*-Messages too.

add_received(*sequence_num: int*) → None
Add received sequence number

Parameters **sequence_num** – Sequence number

Returns: None

add_processed(*sequence_num: int*) → None
Add sequence number of message that was sent to console and succeeded in processing.

Parameters **sequence_num** – Sequence number

Returns: None

add_rejected(*sequence_num: int*) → None
Add sequence number of message that was sent to console and was rejected by it.

Parameters **sequence_num** – Sequence number

Returns: None

next_sequence_num() → int
Get next sequence number to use for outbound *Message*.

Returns: None

property low_watermark
Get current *Low Watermark*

Returns: Low Watermark

exception xbox.sg.protocol.**ChannelError**
Bases: Exception

Exception thrown by *ChannelManager*.

class xbox.sg.protocol.**ChannelManager**
Bases: object

CHANNEL_CORE = 0

CHANNEL_ACK = 11529215046068469760

__init__()
Keep track of established ServiceChannels

handle_channel_start_response(*msg: xbox.sg.utils.struct.XStruct*) → xbox.sg.enum.ServiceChannel
Handle message of type *StartChannelResponse*

Parameters **msg** – Start Channel Response message

Raises ***ChannelError*** – If channel acquire failed

Returns: Acquired ServiceChannel

get_next_request_id(*channel: xbox.sg.enum.ServiceChannel*) → int
Get next Channel request id for ServiceChannel

Incremented on each call.

Parameters **channel** – Service channel

Returns: Channel request id

get_channel(*channel_id: int*) → xbox.sg.enum.ServiceChannel
Get matching ServiceChannel enum for provided Channel ID of *Message*

Parameters **channel_id** – Channel of Message

Returns: Service channel

get_channel_id(*channel: xbox.sg.enum.ServiceChannel*) → int
Get Channel ID for use in *Message* for provided ServiceChannel

Parameters **channel** – Service channel

Returns: Channel ID for use in *Message*

reset() → None
Erase the channels table

Returns None

exception xbox.sg.protocol.**FragmentError**
Bases: Exception

Exception thrown by *FragmentManager*.

class xbox.sg.protocol.**FragmentManager**
Bases: object

Assembles fragmented messages

reassemble_message(*msg: xbox.sg.utils.struct.XStruct*) → Optional[xbox.sg.utils.struct.XStruct]
Reassemble message fragment

Parameters **msg** – Message fragment

Returns: Reassembled / decoded payload on success, *None* if payload is not ready or assembly failed.

reassemble_json(*json_msg: dict*) → Optional[dict]
Reassemble fragmented json message

Parameters **json_msg** – Fragmented json message

Returns: Reassembled / Decoded json object on success, *None* if datagram is not ready or assembly failed

static _encode(*obj: dict*) → str
Dump a dict as json string, then encode with base64

Parameters **obj** – Dict to encode

Returns: base64 encoded string

static _decode(*data: str*) → dict
Decode a base64 encoded json object

Parameters **data** – Base64 string

Returns: Decoded json object

xbox.sg.protocol.**_fragment_connect_request**(*crypto_instance: xbox.sg.crypto.Crypto*, *client_uuid: uuid.UUID*, *pubkey_type: xbox.sg.enum.PublicKeyType*, *pubkey: bytes*, *userhash: str*, *auth_token: str*, *request_num: int = 0*) → List
Internal method to fragment ConnectRequest.

Parameters

- **crypto_instance** – Instance of Crypto
- **client_uuid** – Client UUID
- **Public Key Type** (*pubkey_type*) –
- **pubkey** – Public Key

- **userhash** – Xbox Live Account userhash
- **auth_token** – Xbox Live Account authentication token (XSTS)
- **request_num** – Request Number

Returns List of ConnectRequest fragments

Return type list

CHAPTER

FOUR

CRYPTOGRAPHY

Cryptography portion used for sending Smartglass message

Depending on the foreign public key type, the following Elliptic curves can be used:

- Prime 256R1
- Prime 384R1
- Prime 521R1

1. On Discovery, the console responds with a DiscoveryResponse including a certificate, this certificate holds the console's *public key*.
2. The Client generates appropriate *elliptic curve* and derives the *shared secret* using *console's public key*
3. The *shared secret* is salted via 2x hashes, see *kdf_salts*
4. The *salted shared secret* is hashed using *SHA-512*
5. The *salted & hashed shared secret* is split into the following individual keys:
   - bytes 0-16: Encryption key (AES 128-CBC)
   - bytes 16-32: Initialization Vector key
   - bytes 32-64: Hashing key (HMAC SHA-256)
6. The resulting *public key* from this `Crypto` context is sent with the ConnectRequest message to the console

class xbox.sg.crypto.**SaltType**

Bases: `object`

Define whether Salt is pre- or appended

Prepend = 1

Append = 2

class xbox.sg.crypto.**Salt**(*value*, *salt_type=1*)

Bases: `object`

__init__(*value*, *salt_type=1*)

Handle salting of ECDH shared secret

Parameters

- **value** (`bytes`) – Salting bytes
- **salt_type** (`SaltType`) – Salt Type

apply(*data*)

Appends or prepends salt bytes to data

Parameters **data** (*bytes*) – Data to be salted

Returns Salted data

Return type bytes

class xbox.sg.crypto.**Crypto**(*foreign_public_key*, *privkey=None*, *pubkey=None*)

Bases: object

__init__(*foreign_public_key*, *privkey=None*, *pubkey=None*)

Initialize Crypto context via the foreign public key of the console. The public key is part of the console certificate.

Parameters

- **foreign_public_key** (ec.EllipticCurvePublicKey) – The console's public key
- **privkey** (ec.EllipticCurvePrivateKey) – Optional private key
- **pubkey** (ec.EllipticCurvePublicKey) – Optional public key

property shared_secret

Shared secret

Returns Shared secret

Return type bytes

property pubkey_type

Public Key Type aka. keystrength

Returns Public Key Type

Return type PublicKeyType

property pubkey_bytes

Public Key Bytes (minus the first identifier byte!)

Returns Public key

Return type bytes

property foreign_pubkey

Foreign key that was used to generate this crypto context

Returns Console's public key

Return type ec.EllipticCurvePublicKey

classmethod from_bytes(*foreign_public_key*, *public_key_type=None*)

Initialize Crypto context with foreign public key in bytes / hexstring format.

Parameters

- **foreign_public_key** (*bytes*) – Console's public key
- **public_key_type** (PublicKeyType) – Public Key Type

Returns Instance

Return type *Crypto*

classmethod from_shared_secret(*shared_secret*)

Set up crypto context with shared secret

Parameters **shared_secret** (*bytes*) – The shared secret

Returns Instance

Return type *Crypto*

generate_iv(*seed=None*)
Generates an IV to be used in encryption/decryption

Parameters **seed** (*bytes*) – An optional IV seed

Returns Initialization Vector

Return type bytes

encrypt(*iv*, *plaintext*)
Encrypts plaintext with AES-128-CBC

No padding is added here, data has to be aligned to block size (16 bytes).

Parameters

- **iv** (*bytes*) – The IV to use. None where no IV is used.
- **plaintext** (*bytes*) – The plaintext to encrypt.

Returns Encrypted Data

Return type bytes

decrypt(*iv*, *ciphertext*)
Decrypts ciphertext

No padding is removed here.

Parameters

- **iv** (*bytes*) – The IV to use. None where no IV is used.
- **ciphertext** (*bytes*) – The hex representation of a ciphertext to be decrypted

Returns Decrypted data

Return type bytes

hash(*data*)
Securely hashes data with HMAC SHA-256

Parameters **data** (*bytes*) – The data to securely hash.

Returns Hashed data

Return type bytes

verify(*data*, *secure_hash*)
Verifies that the given data generates the given secure_hash

Parameters

- **data** (*bytes*) – The data to validate.
- **secure_hash** (*bytes*) – The secure hash to validate against.

Returns True on success, False otherwise

Return type bool

class xbox.sg.crypto.**Padding**
Bases: object

Padding base class.

static size(*length*, *alignment*)
Calculate needed padding size.

Parameters

- **length** (*int*) – Data size
- **alignment** (*int*) – Data alignment

Returns Padding size

Return type int

static pad(*payload*, *alignment*)
Abstract method to override

Parameters

- **payload** (*bytes*) – Data blob
- **alignment** (*int*) – Data alignment

Returns Data with padding bytes

Return type bytes

static remove(*payload*)
Common method for removing padding from data blob.

Parameters **payload** (*bytes*) – Padded data.

Returns Data with padding bytes removed

Return type bytes

class xbox.sg.crypto.**PKCS7Padding**
Bases: *xbox.sg.crypto.Padding*

static pad(*payload*, *alignment*)
Add PKCS#7 padding to data blob.

Parameters

- **payload** (*bytes*) – Data blob
- **alignment** (*int*) – Data alignment

Returns Data with padding bytes

Return type bytes

class xbox.sg.crypto.**ANSIX923Padding**
Bases: *xbox.sg.crypto.Padding*

static pad(*payload*, *alignment*)
Add ANSI.X923 padding to data blob.

Parameters

- **payload** (*bytes*) – Data blob
- **alignment** (*int*) – Data alignment

Returns Data with padding bytes

Return type bytes

CHAPTER

FIVE

PACKET FACTORY - ASSEMBLE PACKETS

Smartglass packet factory

xbox.sg.factory.**_message_header**(*msg_type*, *channel_id=0*, *target_participant_id=0*, *source_participant_id=0*, *is_fragment=False*, *need_ack=False*)

Helper method for creating a message header.

Parameters

- **msg_type** (*int*) – The message type.
- **channel_id** (*int*) – The target channel of the message.
- **target_participant_id** (*int*) – The target participant Id.
- **source_participant_id** (*int*) – The source participant Id.
- **is_fragment** (*bool*) – Whether the message is a fragment.
- **need_ack** (*bool*) – Whether the message needs an acknowledge.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**power_on**(*liveid*)

Assemble PowerOn Request message.

Parameters **liveid** (*str*) – The console LiveId (extracted from the DiscoveryResponse Certificate).

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**discovery**(*client_type=<ClientType.Android: 8>*)

Assemble DiscoveryRequest SimpleMessage.

Parameters **client_type** (ClientType) – Member of ClientType, defaults to *ClientType.Android*.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**connect**(*sg_uuid*, *public_key_type*, *public_key*, *iv*, *userhash*, *jwt*, *msg_num*, *num_start*, *num_end*)

Assemble ConnectRequest SimpleMessage.

Parameters

- **sg_uuid** (*UUID*) – Client Uuid, randomly generated.

- **public_key_type** (PublicKeyType) – Public Key Type.
- **public_key** (*bytes*) – Calculated Public Key, from Crypto.
- **iv** (*bytes*) – Initialization Vector for this encrypted message.
- **userhash** (*str*) – Xbox Live Account userhash.
- **jwt** (*str*) – Xbox Live Account JWT / Auth-token.
- **msg_num** (*int*) – Current message number (important for fragmentation).
- **num_start** (*int*) – Base number start of ConnectRequest fragments.
- **num_end** (*int*) – Base number end of ConnectRequest fragments (base number start + total fragments + 1).

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**message_fragment**(*msg_type*, *sequence_begin*, *sequence_end*, *data*, ***kwargs*)
Assemble fragmented message.

Parameters

- **msg_type** (*int*) – Base Message Type.
- **sequence_begin** (*int*) – Sequence number with first fragment.
- **sequence_end** (*int*) – Last sequence number (+1) containing fragment.
- **data** (*bytes*) – Plaintext MessagePacket payload fragment.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**acknowledge**(*low_watermark*, *processed_list*, *rejected_list*, ***kwargs*)
Assemble acknowledgement message.

Parameters

- **low_watermark** (*int*) – Low Watermark.
- **processed_list** (*list*) – List of processed message sequence numbers.
- **rejected_list** (*list*) – List of rejected message sequence numbers.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**json**(*text*, ***kwargs*)
Assemble JSON message.

Parameters **text** (*str*) – Text string.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**disconnect**(*reason*, *error_code*, ***kwargs*)
Assemble Disconnect message.

Parameters

- **reason** (xbox.sg.enum.DisconnectReason) – Disconnect reason.
- **error_code** (*int*) – Error code.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**power_off**(*liveid*, ***kwargs*)
Assemble PowerOff message.

Parameters **liveid** (*str*) – Live ID of console.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**local_join**(*client_info*, ***kwargs*)
Assemble LocalJoin message.

Parameters **client_info** (*object*) – Instance of WindowsClientInfo or AndroidClientInfo.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**title_launch**(*location*, *uri*, ***kwargs*)
Assemble TitleLaunch message.

Parameters

- **location** (ActiveTitleLocation) – Location.
- **uri** (*str*) – Uri string for title to launch.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**start_channel**(*channel_request_id*, *title_id*, *service*, *activity_id*, ***kwargs*)
Assemble StartChannelRequest message.

Parameters

- **channel_request_id** (*int*) – Incrementing Channel Request Id.
- **title_id** (*int*) – Title Id, usually 0.
- **service** (MessageTarget) – Member of MessageTarget.
- **activity_id** (*int*) – Activity Id, usually 0.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**stop_channel**(*channel_id*, ***kwargs*)
Assemble StopChannel message.

Parameters **channel_id** (*int*) – Channel Id to stop.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**gamepad**(*timestamp*, *buttons*, *l_trigger*, *r_trigger*, *l_thumb_x*, *l_thumb_y*, *r_thumb_x*, *r_thumb_y*, ***kwargs*)
Assemble gamepad input message.

Parameters

- **timestamp** (*longlong*) – Timestamp.

- **buttons** (GamePadButton) – Bitmask of pressed gamepad buttons.
- **l_trigger** (*float*) – LT.
- **r_trigger** (*float*) – RT.
- **l_thumb_x** (*float*) – Position of left thumbstick, X-Axis.
- **l_thumb_y** (*float*) – Position of left thumbstick, Y-Axis.
- **r_thumb_x** (*float*) – Position of right thumbstick, X-Axis.
- **r_thumb_y** (*float*) – Position of right thumbstick, Y-Axis.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**unsnap**(*unknown*, ***kwargs*)
Assemble unsnap message.

Parameters **unknown** (*int*) – Unknown value.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**game_dvr_record**(*start_time_delta*, *end_time_delta*, ***kwargs*)
Assemble Game DVR record message.

Parameters

- **start_time_delta** (*int*) – Start Time delta.
- **end_time_delta** (*int*) – End Time delta.

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**media_command**(*request_id*, *title_id*, *command*, *seek_position*, ***kwargs*)
Assemble Media Command message.

Parameters

- **request_id** (*int*) – Request Id of MediaCommand.
- **title_id** (*int*) – Title Id of Application to control.
- **command** (MediaControlCommand) – Media Command.
- **seek_position** – Seek position.

xbox.sg.factory.**systemtext_input**(*session_id*, *base_version*, *submitted_version*, *total_text_len*, *selection_start*, *selection_length*, *flags*, *text_chunk_byte_start*, *text_chunk*, *delta=None*, ***kwargs*)
Assemble SystemText Input message

Parameters

- **session_id** (*int*) – Textt session Id
- **base_version** (*int*) – Base version
- **submitted_version** (*int*) – Submitted Version
- **total_text_len** (*int*) – Total text length
- **selection_start** (*int*) – Selection start

- **selection_length** (*int*) – Selection length
- **flags** (*int*) – Input flags
- **text_chunk_byte_start** (*int*) – Start byte of text chunk
- **text_chunk** (*str*) – Actual text to send
- **delta** (*NoneType*) – Unknown

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**systemtext_ack**(*session_id*, *text_version*, ***kwargs*)
Assemble SystemText Acknowledge message

Parameters

- **session_id** (*int*) – Text session Id
- **text_version** (*int*) – Text version to acknowledge

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**systemtext_done**(*session_id*, *text_version*, *flags*, *result*, ***kwargs*)
Assemble SystemText Done message

Parameters

- **session_id** (*int*) – Text session Id
- **text_version** (*int*) – Text version
- **flags** (*int*) – Flags
- **result** (TextResult) – Text result

Returns Instance of :class:`XStructObj`.

Return type XStructObj

xbox.sg.factory.**title_auxiliary_stream**(***kwargs*)
Assemble Auxiliary Stream message

Returns Instance of XStructObj

Return type XStructObj

CHAPTER

SIX

XBOX.AUXILIARY PACKAGE

6.1 Subpackages

6.1.1 xbox.auxiliary.scripts package

Submodules

xbox.auxiliary.scripts.fo4 module

Module contents

6.2 Submodules

6.2.1 xbox.auxiliary.crypto module

Cryptography portion used for Title Channel aka Auxiliary Stream

class xbox.auxiliary.crypto.**AuxiliaryStreamCrypto**(*crypto_key*, *hash_key*, *server_iv*, *client_iv*)

Bases: object

__init__(*crypto_key*, *hash_key*, *server_iv*, *client_iv*)
Initialize Auxiliary Stream Crypto-context.

classmethod from_connection_info(*connection_info*)
Initialize Crypto context via AuxiliaryStream-message connection info.

encrypt(*plaintext*)
Encrypts plaintext with AES-128-CBC

No padding is added here, data has to be aligned to block size (16 bytes).

Parameters **plaintext** (*bytes*) – The plaintext to encrypt.

Returns Encrypted Data

Return type bytes

encrypt_server(*plaintext*)

decrypt(*ciphertext*)
Decrypts ciphertext

No padding is removed here.

Parameters **ciphertext** (*bytes*) – Ciphertext to be decrypted

Returns Decrypted data

Return type bytes

decrypt_client(*ciphertext*)

hash(*data*)
Securely hashes data with HMAC SHA-256

Parameters **data** (*bytes*) – The data to securely hash.

Returns Hashed data

Return type bytes

verify(*data*, *secure_hash*)
Verifies that the given data generates the given secure_hash

Parameters

- **data** (*bytes*) – The data to validate.
- **secure_hash** (*bytes*) – The secure hash to validate against.

Returns True on success, False otherwise

Return type bool

6.2.2 xbox.auxiliary.manager module

exception xbox.auxiliary.manager.**TitleManagerError**
Bases: Exception

Exception thrown by TitleManager

class xbox.auxiliary.manager.**TitleManager**(*console*)
Bases: *xbox.sg.manager.Manager*

__init__(*console*)
Title Manager (ServiceChannel.Title)

Parameters **console** (*Console*) – Console object, internally passed by *Console.add_manager*

_on_message(*msg*, *channel*)
Internal handler method to receive messages from Title Channel

Parameters

- **msg** (XStructObj) – Message
- **channel** (ServiceChannel) – Service channel

async start_title_channel(*title_id: int*) → Any

property active_surface
Get *Active Surface*.

Returns Active Surface

Return type XStructObj

property connection_info
Get current *Connection info*

Returns Connection info

Return type XStructObj

6.2.3 xbox.auxiliary.packer module

exception xbox.auxiliary.packer.**AuxiliaryPackerException**
Bases: Exception

xbox.auxiliary.packer.**pack**(*data: bytes*, *crypto: xbox.auxiliary.crypto.AuxiliaryStreamCrypto*, *server_data: bool = False*) → List[bytes]
Encrypt auxiliary data blob

Parameters

- **data** – Data
- **crypto** – Crypto context
- **server_data** – Whether to encrypt with *server IV*

Returns Encrypted message

Return type bytes

xbox.auxiliary.packer.**unpack**(*data: bytes*, *crypto: xbox.auxiliary.crypto.AuxiliaryStreamCrypto*, *client_data: bool = False*) → bytes
Split and decrypt auxiliary data blob

Parameters

- **data** – Data blob
- **crypto** – Crypto context
- **client_data** – Whether to decrypt with 'client IV'

Returns Decrypted message

Return type bytes

6.2.4 xbox.auxiliary.packet module

6.2.5 xbox.auxiliary.relay module

exception xbox.auxiliary.relay.**AuxiliaryPackerException**
Bases: Exception

class xbox.auxiliary.relay.**ConsoleConnection**(*address*, *port*, *crypto*)
Bases: object

BUFFER_SIZE = 2048

start()

stop()

handle(*data*)

async send(*msg*)

class xbox.auxiliary.relay.**LocalConnection**
Bases: asyncio.protocols.Protocol

data_received_event = <xbox.sg.utils.events.Event object>

connection_made_event = <xbox.sg.utils.events.Event object>

connection_made(*transport: asyncio.transports.BaseTransport*) → None
Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data_received() calls. When the connection is closed, connection_lost() is called.

data_received(*data: bytes*) → None
Called when some data is received.

The argument is a bytes object.

close_connection() → None

class xbox.auxiliary.relay.**AuxiliaryRelayService**(*loop: asyncio.events.AbstractEventLoop*, *connection_info: xbox.sg.utils.struct.XStructObj*, *listen_port: int*)

Bases: object

async run()

connection_made(*transport*)

6.3 Module contents

Auxiliary stream support for smartglass

CHAPTER

SEVEN

XBOX.STUMP PACKAGE

7.1 Submodules

7.1.1 xbox.stump.enum module

Stump enumerations

class xbox.stump.enum.**Message**(*value*)
Bases: str, enum.Enum

Message types

ERROR = 'Error'

ENSURE_STREAMING_STARTED = 'EnsureStreamingStarted'

CONFIGURATION = 'GetConfiguration'

HEADEND_INFO = 'GetHeadendInfo'

LIVETV_INFO = 'GetLiveTVInfo'

PROGRAMM_INFO = 'GetProgrammInfo'

RECENT_CHANNELS = 'GetRecentChannels'

TUNER_LINEUPS = 'GetTunerLineups'

APPCHANNEL_DATA = 'GetAppChannelData'

APPCHANNEL_LINEUPS = 'GetAppChannelLineups'

APPCHANNEL_PROGRAM_DATA = 'GetAppChannelProgramData'

SEND_KEY = 'SendKey'

SET_CHANNEL = 'SetChannel'

class xbox.stump.enum.**Notification**(*value*)
Bases: str, enum.Enum

Notification types

STREAMING_ERROR = 'StreamingError'

CHANNEL_CHANGED = 'ChannelChanged'

CHANNELTYPE_CHANGED = 'ChannelTypeChanged'

CONFIGURATION_CHANGED = 'ConfigurationChanged'

DEVICE_UI_CHANGED = 'DeviceUIChanged'

HEADEND_CHANGED = 'HeadendChanged'

VIDEOFORMAT_CHANGED = 'VideoFormatChanged'

PROGRAM_CHANGED = 'ProgrammChanged'

TUNERSTATE_CHANGED = 'TunerStateChanged'

class xbox.stump.enum.**Source**(*value*)
Bases: str, enum.Enum

Streamingsources

HDMI = 'hdmi'

TUNER = 'tuner'

class xbox.stump.enum.**DeviceType**(*value*)
Bases: str, enum.Enum

Devicetypes

TV = 'tv'

TUNER = 'tuner'

SET_TOP_BOX = 'stb'

AV_RECEIVER = 'avr'

class xbox.stump.enum.**SourceHttpQuery**(*value*)
Bases: str, enum.Enum

Source strings used in HTTP query

HDMI = 'hdmi-in'

TUNER = 'zurich'

class xbox.stump.enum.**Input**
Bases: object

HDMI = UUID('ba5eba11-dea1-4bad-ba11-feddeadfab1e')

class xbox.stump.enum.**Quality**(*value*)
Bases: str, enum.Enum

Quality values

LOW = 'low'

MEDIUM = 'medium'

HIGH = 'high'

BEST = 'best'

class xbox.stump.enum.**FilterType**(*value*)
Bases: str, enum.Enum

Channel-filter types

ALL = 'ALL'

HDSD = 'HDSD'

HD = 'HD'

7.1.2 xbox.stump.json_model module

JSON models for deserializing Stump messages

exception xbox.stump.json_model.**StumpJsonError**
Bases: Exception

class xbox.stump.json_model.**StumpRequest**(*, *msgid: str*, *request: str*, *params: dict = None*)
Bases: pydantic.main.BaseModel

msgid: **str**

request: **str**

params: **Optional[dict]**

class xbox.stump.json_model.**StumpResponse**(*, *msgid: str*, *response: str*)
Bases: pydantic.main.BaseModel

msgid: **str**

response: **str**

class xbox.stump.json_model.**StumpError**(*, *msgid: str*, *error: str*)
Bases: pydantic.main.BaseModel

msgid: **str**

error: **str**

class xbox.stump.json_model.**StumpNotification**(*, *notification: str*)
Bases: pydantic.main.BaseModel

notification: **str**

class xbox.stump.json_model.**AppChannelLineups**(*, *msgid: str*, *response: str*, *params: List[xbox.stump.json_model._AppProvider]*)
Bases: *xbox.stump.json_model.StumpResponse*

params: **List[xbox.stump.json_model._AppProvider]**

class xbox.stump.json_model.**EnsureStreamingStarted**(*, *msgid: str*, *response: str*, *params: xbox.stump.json_model._EnsureStreamingStarted*)
Bases: *xbox.stump.json_model.StumpResponse*

params: **xbox.stump.json_model._EnsureStreamingStarted**

class xbox.stump.json_model.**TunerLineups**(*, *msgid: str*, *response: str*, *params: xbox.stump.json_model._TunerLineups*)
Bases: *xbox.stump.json_model.StumpResponse*

params: **xbox.stump.json_model._TunerLineups**

class xbox.stump.json_model.**SendKey**(*, *msgid: str*, *response: str*, *params: bool*)
Bases: *xbox.stump.json_model.StumpResponse*

params: **bool**

class xbox.stump.json_model.**RecentChannels**(*, *msgid: str*, *response: str*, *params: List[xbox.stump.json_model._RecentChannel]*)
Bases: *xbox.stump.json_model.StumpResponse*

params: **List[xbox.stump.json_model._RecentChannel]**

class xbox.stump.json_model.**Configuration**(*, *msgid: str*, *response: str*, *params: List[xbox.stump.json_model._DeviceConfiguration]*)

Bases: *xbox.stump.json_model.StumpResponse*

params: List[xbox.stump.json_model._DeviceConfiguration]

class xbox.stump.json_model.**LiveTvInfo**(*, *msgid: str*, *response: str*, *params: xbox.stump.json_model._LiveTvInfo*)

Bases: *xbox.stump.json_model.StumpResponse*

params: xbox.stump.json_model._LiveTvInfo

class xbox.stump.json_model.**HeadendInfo**(*, *msgid: str*, *response: str*, *params: xbox.stump.json_model._HeadendInfo*)

Bases: *xbox.stump.json_model.StumpResponse*

params: xbox.stump.json_model._HeadendInfo

xbox.stump.json_model.**deserialize_stump_message**(*data: dict*) → Union[*xbox.stump.json_model.StumpError*, *xbox.stump.json_model.StumpNotification*, *xbox.stump.json_model.StumpResponse*]

Helper for deserializing JSON stump messages

Parameters **data** (*dict*) – Stump message

Returns Parsed JSON object

Return type Model

7.1.3 xbox.stump.manager module

StumpManager - Handling TV Streaming / IR control commands

exception xbox.stump.manager.**StumpException**

Bases: Exception

Exception thrown by StumpManager

class xbox.stump.manager.**StumpManager**(*console*)

Bases: *xbox.sg.manager.Manager*

__init__(*console*)

Title Manager (ServiceChannel.SystemInputTVRemote)

Parameters **console** – Console object

property msg_id

Internal method for generating message IDs.

Returns A new message ID.

Return type str

generate_stream_url(*source: xbox.stump.enum.Source*, *xuid: str = None*, *quality: xbox.stump.enum.Quality = <Quality.BEST: 'best'>*) → str

Generate TV Streaming URL (Dvb-USB tuner, not hdmi-in)

Parameters

- **source** – Streaming source
- **xuid** – optional, XUID
- **quality** – Streaming quality

Returns: HTTP URL

_on_json(*data: dict*, *channel: int*) → None
Internal handler for JSON messages received by the core protocol.

Parameters

- **data** – The JSON object that was received.
- **channel** – The channel this message was received on.

Returns None.

_on_response(*message_type: Union[xbox.stump.enum.Message, str]*, *data: xbox.stump.json_model.StumpResponse*) → None
Internal response handler. For logging purposes.

Parameters

- **message_type** – The message type.
- **data** – The raw message.

_on_notification(*notification: Union[xbox.stump.enum.Notification, str]*, *data: xbox.stump.json_model.StumpNotification*) → None
Internal notification handler. For logging purposes.

Parameters

- **notification** – The notification type.
- **data** – The raw message.

_on_error(*data: xbox.stump.json_model.StumpError*) → None
Internal error handler.

Parameters **data** – The error dictionary from the Message.

async _send_stump_message(*name*, *params=None*, *msgid=None*, *timeout=3*)
Internal method for sending JSON messages over the core protocol.

Handles message IDs as well as waiting for results.

Parameters

- **name** (*Enum*) – Request name
- **params** (*dict*) – The message parameters to send.
- **msgid** (*str*) – Message identifier
- **timeout** (*int*) – Timeout in seconds

Returns The received result.

Return type dict

async request_stump_configuration() → dict
Request device configuration from console.

The configuration holds info about configured, by Xbox controlable devices (TV, AV, STB).

Returns The received result.

Return type dict

async request_headend_info() → dict
Request available headend information from console.

Returns: The received result.

async request_live_tv_info() → dict
Request LiveTV information from console.

Holds information about currently tuned channel, streaming-port etc.

Returns: The received result.

async request_program_info() → dict
Request program information.

NOTE: Not working?!

Returns: The received result.

async request_tuner_lineups() → dict
Request Tuner Lineups from console.

Tuner lineups hold information about scanned / found channels.

Returns: The received result.

async request_app_channel_lineups() → dict
Request AppChannel Lineups.

Returns: The received result.

async request_app_channel_data(*provider_id: str*, *channel_id: str*) → dict
Request AppChannel Data.

Parameters

- **provider_id** – Provider ID.
- **channel_id** – Channel ID.

Returns: The received result.

async request_app_channel_program_data(*provider_id: str*, *program_id: str*) → dict
Request AppChannel Program Data.

Parameters

- **provider_id** – Provider ID.
- **program_id** – Program Id.

Returns: The received result.

async set_stump_channel_by_id(*channel_id: str*, *lineup_id: str*) → dict
Switch to channel by providing channel ID and lineup ID.

Parameters

- **channel_id** – Channel ID.
- **lineup_id** – Lineup ID.

Returns: The received result.

async set_stump_channel_by_name(*channel_name: str*) → dict
Switch to channel by providing channel name.

Parameters **channel_name** – Channel name to switch to.

Returns: The received result.

async request_recent_channels(*first: int*, *count: int*) → dict
Request a list of recently watched channels.

Parameters

- **first** – Where to start enumeration.
- **count** – Number of channels to request.

Returns: The received result.

async send_stump_key(*button: str*, *device_id: str = None*, ***kwargs*) → dict
Send a remote control button to configured device via Xbox's IR Blaster / Kinect / whatev.

Parameters

- **button** – Button to send.
- **device_id** – Device ID of device to control.

Returns: The received result.

async request_ensure_stump_streaming_started(*source: xbox.stump.enum.Source*) → dict
Ensure that streaming started on desired tuner type

Parameters **source** – Tuner Source to check for.

Returns: The received result.

property headend_locale

property streaming_port

property is_hdmi_mode

property current_tunerchannel_type

7.2 Module contents

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

xbox.auxiliary, 34
xbox.auxiliary.crypto, 31
xbox.auxiliary.manager, 32
xbox.auxiliary.packer, 33
xbox.auxiliary.packet, 33
xbox.auxiliary.relay, 33
xbox.sg.console, 1
xbox.sg.crypto, 21
xbox.sg.factory, 25
xbox.sg.manager, 7
xbox.sg.protocol, 13
xbox.stump, 41
xbox.stump.enum, 35
xbox.stump.json_model, 37
xbox.stump.manager, 38

INDEX

Symbols

__init__() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 31
__init__() (*xbox.auxiliary.manager.TitleManager method*), 32
__init__() (*xbox.sg.console.Console method*), 1
__init__() (*xbox.sg.crypto.Crypto method*), 22
__init__() (*xbox.sg.crypto.Salt method*), 21
__init__() (*xbox.sg.manager.InputManager method*), 8
__init__() (*xbox.sg.manager.Manager method*), 7
__init__() (*xbox.sg.manager.MediaManager method*), 9
__init__() (*xbox.sg.manager.TextManager method*), 10
__init__() (*xbox.sg.protocol.ChannelManager method*), 18
__init__() (*xbox.sg.protocol.SequenceManager method*), 17
__init__() (*xbox.sg.protocol.SmartglassProtocol method*), 13
__init__() (*xbox.stump.manager.StumpManager method*), 38
_await_ack() (*xbox.sg.protocol.SmartglassProtocol method*), 14
_decode() (*xbox.sg.protocol.FragmentManager static method*), 19
_encode() (*xbox.sg.protocol.FragmentManager static method*), 19
_ensure_global_protocol_started() (*xbox.sg.console.Console class method*), 2
_ensure_protocol_started() (*xbox.sg.console.Console method*), 2
_fragment_connect_request() (*in module xbox.sg.protocol*), 19
_handle_json() (*xbox.sg.console.Console method*), 4
_handle_message() (*xbox.sg.console.Console method*), 4
_handle_timeout() (*xbox.sg.console.Console method*), 4
_heartbeat_task() (*xbox.sg.protocol.SmartglassProtocol method*), 14
_message_header() (*in module xbox.sg.factory*), 25
_on_ack() (*xbox.sg.protocol.SmartglassProtocol method*), 15
_on_error() (*xbox.stump.manager.StumpManager method*), 39
_on_json() (*xbox.sg.manager.Manager method*), 8
_on_json() (*xbox.sg.protocol.SmartglassProtocol method*), 15
_on_json() (*xbox.stump.manager.StumpManager method*), 39
_on_message() (*xbox.auxiliary.manager.TitleManager method*), 32
_on_message() (*xbox.sg.manager.InputManager method*), 8
_on_message() (*xbox.sg.manager.Manager method*), 8
_on_message() (*xbox.sg.manager.MediaManager method*), 9
_on_message() (*xbox.sg.manager.TextManager method*), 10
_on_message() (*xbox.sg.protocol.SmartglassProtocol method*), 15
_on_notification() (*xbox.stump.manager.StumpManager method*), 39
_on_response() (*xbox.stump.manager.StumpManager method*), 39
_reset_state() (*xbox.sg.console.Console method*), 4
_send() (*xbox.sg.protocol.SmartglassProtocol method*), 14
_send_json() (*xbox.sg.manager.Manager method*), 8
_send_message() (*xbox.sg.manager.Manager method*), 8
_send_stump_message() (*xbox.stump.manager.StumpManager method*), 39
_set_result() (*xbox.sg.protocol.SmartglassProtocol method*), 14

A

ack() (*xbox.sg.protocol.SmartglassProtocol method*), 16
acknowledge() (*in module xbox.sg.factory*), 26
active_media() (*xbox.sg.manager.MediaManager property*), 9
active_surface() (*xbox.auxiliary.manager.TitleManager property*), 32
active_surface() (*xbox.sg.console.Console property*), 5
add_manager() (*xbox.sg.console.Console method*), 2
add_processed() (*xbox.sg.protocol.SequenceManager method*), 17
add_received() (*xbox.sg.protocol.SequenceManager method*), 17
add_rejected() (*xbox.sg.protocol.SequenceManager method*), 18
ALL (*xbox.stump.enum.FilterType attribute*), 36
anonymous_connection_allowed() (*xbox.sg.console.Console property*), 5
ANSIX923Padding (*class in xbox.sg.crypto*), 24
APPCHANNEL_DATA (*xbox.stump.enum.Message attribute*), 35
APPCHANNEL_LINEUPS (*xbox.stump.enum.Message attribute*), 35
APPCHANNEL_PROGRAM_DATA (*xbox.stump.enum.Message attribute*), 35
AppChannelLineups (*class in xbox.stump.json_model*), 37
Append (*xbox.sg.crypto.SaltType attribute*), 21
apply() (*xbox.sg.crypto.Salt method*), 21
asset_id() (*xbox.sg.manager.MediaManager property*), 9
aum_id() (*xbox.sg.manager.MediaManager property*), 9
authenticated_users_allowed() (*xbox.sg.console.Console property*), 5
AuxiliaryPackerException, 33
AuxiliaryRelayService (*class in xbox.auxiliary.relay*), 34
AuxiliaryStreamCrypto (*class in xbox.auxiliary.crypto*), 31
AV_RECEIVER (*xbox.stump.enum.DeviceType attribute*), 36
available() (*xbox.sg.console.Console property*), 5

B

BEST (*xbox.stump.enum.Quality attribute*), 36
BUFFER_SIZE (*xbox.auxiliary.relay.ConsoleConnection attribute*), 33

C

CHANNEL_ACK (*xbox.sg.protocol.ChannelManager attribute*), 18
CHANNEL_CHANGED (*xbox.stump.enum.Notification attribute*), 35
CHANNEL_CORE (*xbox.sg.protocol.ChannelManager attribute*), 18
ChannelError, 18
ChannelManager (*class in xbox.sg.protocol*), 18
CHANNELTYPE_CHANGED (*xbox.stump.enum.Notification attribute*), 35
close_connection() (*xbox.auxiliary.relay.LocalConnection method*), 34
Configuration (*class in xbox.stump.json_model*), 37
CONFIGURATION (*xbox.stump.enum.Message attribute*), 35
CONFIGURATION_CHANGED (*xbox.stump.enum.Notification attribute*), 35
connect() (*in module xbox.sg.factory*), 25
connect() (*xbox.sg.console.Console method*), 3
connect() (*xbox.sg.protocol.SmartglassProtocol method*), 15
connected() (*xbox.sg.console.Console property*), 5
connection_info() (*xbox.auxiliary.manager.TitleManager property*), 32
connection_lost() (*xbox.sg.protocol.SmartglassProtocol method*), 13
connection_made() (*xbox.auxiliary.relay.AuxiliaryRelayService method*), 34
connection_made() (*xbox.auxiliary.relay.LocalConnection method*), 34
connection_made() (*xbox.sg.protocol.SmartglassProtocol method*), 13
connection_made_event (*xbox.auxiliary.relay.LocalConnection attribute*), 34
connection_state() (*xbox.sg.console.Console property*), 5
Console (*class in xbox.sg.console*), 1
console_status() (*xbox.sg.console.Console property*), 5
console_users_allowed() (*xbox.sg.console.Console property*), 5
ConsoleConnection (*class in xbox.auxiliary.relay*), 33
Crypto (*class in xbox.sg.crypto*), 22
current_text_version() (*xbox.sg.manager.TextManager property*), 11

current_tunerchannel_type() (*xbox.stump.manager.StumpManager property*), 41

D

data_received() (*xbox.auxiliary.relay.LocalConnection method*), 34
data_received_event (*xbox.auxiliary.relay.LocalConnection attribute*), 33
datagram_received() (*xbox.sg.protocol.SmartglassProtocol method*), 14
decrypt() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 31
decrypt() (*xbox.sg.crypto.Crypto method*), 23
decrypt_client() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 32
deserialize_stump_message() (*in module xbox.stump.json_model*), 38
device_status() (*xbox.sg.console.Console property*), 5
DEVICE_UI_CHANGED (*xbox.stump.enum.Notification attribute*), 35
DeviceType (*class in xbox.stump.enum*), 36
disconnect() (*in module xbox.sg.factory*), 26
disconnect() (*xbox.sg.console.Console method*), 4
disconnect() (*xbox.sg.protocol.SmartglassProtocol method*), 17
discover() (*xbox.sg.console.Console class method*), 2
discover() (*xbox.sg.protocol.SmartglassProtocol method*), 15
discovered() (*xbox.sg.console.Console class method*), 3
discovered() (*xbox.sg.protocol.SmartglassProtocol property*), 15
discovery() (*in module xbox.sg.factory*), 25

E

enabled_commands() (*xbox.sg.manager.MediaManager property*), 9
encrypt() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 31
encrypt() (*xbox.sg.crypto.Crypto method*), 23
encrypt_server() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 31
ENSURE_STREAMING_STARTED (*xbox.stump.enum.Message attribute*), 35
EnsureStreamingStarted (*class in xbox.stump.json_model*), 37
ERROR (*xbox.stump.enum.Message attribute*), 35
error (*xbox.stump.json_model.StumpError attribute*), 37
error_received() (*xbox.sg.protocol.SmartglassProtocol method*), 13

F

FilterType (*class in xbox.stump.enum*), 36
finish_text_input() (*xbox.sg.manager.TextManager method*), 11
foreign_pubkey() (*xbox.sg.crypto.Crypto property*), 22
FragmentError, 19
FragmentManager (*class in xbox.sg.protocol*), 19
from_bytes() (*xbox.sg.crypto.Crypto class method*), 22
from_connection_info() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto class method*), 31
from_dict() (*xbox.sg.console.Console class method*), 2
from_message() (*xbox.sg.console.Console class method*), 2
from_shared_secret() (*xbox.sg.crypto.Crypto class method*), 22

G

game_dvr_record() (*in module xbox.sg.factory*), 28
game_dvr_record() (*xbox.sg.console.Console method*), 4
game_dvr_record() (*xbox.sg.protocol.SmartglassProtocol method*), 17
gamepad() (*in module xbox.sg.factory*), 27
gamepad_input() (*xbox.sg.manager.InputManager method*), 8
generate_iv() (*xbox.sg.crypto.Crypto method*), 23
generate_stream_url() (*xbox.stump.manager.StumpManager method*), 38
get_channel() (*xbox.sg.protocol.ChannelManager method*), 18
get_channel_id() (*xbox.sg.protocol.ChannelManager method*), 18
get_next_request_id() (*xbox.sg.protocol.ChannelManager method*), 18
get_active_session() (*xbox.sg.manager.TextManager property*), 10

H

handle() (*xbox.auxiliary.relay.ConsoleConnection method*), 33

handle_channel_start_response() (*xbox.sg.protocol.ChannelManager method*), 18
hash() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 32
hash() (*xbox.sg.crypto.Crypto method*), 23
HD (*xbox.stump.enum.FilterType attribute*), 36
HDMI (*xbox.stump.enum.Input attribute*), 36
HDMI (*xbox.stump.enum.Source attribute*), 36
HDMI (*xbox.stump.enum.SourceHttpQuery attribute*), 36
HDSD (*xbox.stump.enum.FilterType attribute*), 36
HEADEND_CHANGED (*xbox.stump.enum.Notification attribute*), 35
HEADEND_INFO (*xbox.stump.enum.Message attribute*), 35
headend_locale() (*xbox.stump.manager.StumpManager property*), 41
HeadendInfo (*class in xbox.stump.json_model*), 38
HEARTBEAT_INTERVAL (*xbox.sg.protocol.SmartglassProtocol attribute*), 13
HIGH (*xbox.stump.enum.Quality attribute*), 36

I

Input (*class in xbox.stump.enum*), 36
InputManager (*class in xbox.sg.manager*), 8
InputManagerError, 8
is_certificate_pending() (*xbox.sg.console.Console property*), 5
is_hdmi_mode() (*xbox.stump.manager.StumpManager property*), 41

J

json() (*in module xbox.sg.factory*), 26
json() (*xbox.sg.console.Console method*), 3
json() (*xbox.sg.protocol.SmartglassProtocol method*), 16

L

launch_title() (*xbox.sg.console.Console method*), 3
launch_title() (*xbox.sg.protocol.SmartglassProtocol method*), 17
LIVETV_INFO (*xbox.stump.enum.Message attribute*), 35
LiveTvInfo (*class in xbox.stump.json_model*), 38
local_join() (*in module xbox.sg.factory*), 27
local_join() (*xbox.sg.protocol.SmartglassProtocol method*), 16
LocalConnection (*class in xbox.auxiliary.relay*), 33
LOW (*xbox.stump.enum.Quality attribute*), 36
low_watermark() (*xbox.sg.protocol.SequenceManager property*), 18

M

Manager (*class in xbox.sg.manager*), 7
max_seek() (*xbox.sg.manager.MediaManager property*), 10
max_text_length() (*xbox.sg.manager.TextManager property*), 11
media_command() (*in module xbox.sg.factory*), 28
media_command() (*xbox.sg.manager.MediaManager method*), 10
media_end() (*xbox.sg.manager.MediaManager property*), 10
media_start() (*xbox.sg.manager.MediaManager property*), 10
media_state() (*xbox.sg.manager.MediaManager property*), 9
media_type() (*xbox.sg.manager.MediaManager property*), 9
MediaManager (*class in xbox.sg.manager*), 8
MediaManagerError, 8
MEDIUM (*xbox.stump.enum.Quality attribute*), 36
Message (*class in xbox.stump.enum*), 35
message_fragment() (*in module xbox.sg.factory*), 26
metadata() (*xbox.sg.manager.MediaManager property*), 10
min_seek() (*xbox.sg.manager.MediaManager property*), 10
module
    xbox.auxiliary, 34
    xbox.auxiliary.crypto, 31
    xbox.auxiliary.manager, 32
    xbox.auxiliary.packer, 33
    xbox.auxiliary.packet, 33
    xbox.auxiliary.relay, 33
    xbox.sg.console, 1
    xbox.sg.crypto, 21
    xbox.sg.factory, 25
    xbox.sg.manager, 7
    xbox.sg.protocol, 13
    xbox.stump, 41
    xbox.stump.enum, 35
    xbox.stump.json_model, 37
    xbox.stump.manager, 38
msg_id() (*xbox.stump.manager.StumpManager property*), 38
msgid (*xbox.stump.json_model.StumpError attribute*), 37
msgid (*xbox.stump.json_model.StumpRequest attribute*), 37
msgid (*xbox.stump.json_model.StumpResponse attribute*), 37

N

next_sequence_num()

(*xbox.sg.protocol.SequenceManager method*), 18
Notification (*class in xbox.stump.enum*), 35
notification (*xbox.stump.json_model.StumpNotification attribute*), 37

P

pack() (*in module xbox.auxiliary.packer*), 33
pad() (*xbox.sg.crypto.ANSIX923Padding static method*), 24
pad() (*xbox.sg.crypto.Padding static method*), 24
pad() (*xbox.sg.crypto.PKCS7Padding static method*), 24
Padding (*class in xbox.sg.crypto*), 23
paired() (*xbox.sg.console.Console property*), 5
pairing_state() (*xbox.sg.console.Console property*), 5
params (*xbox.stump.json_model.AppChannelLineups attribute*), 37
params (*xbox.stump.json_model.Configuration attribute*), 38
params (*xbox.stump.json_model.EnsureStreamingStarted attribute*), 37
params (*xbox.stump.json_model.HeadendInfo attribute*), 38
params (*xbox.stump.json_model.LiveTvInfo attribute*), 38
params (*xbox.stump.json_model.RecentChannels attribute*), 37
params (*xbox.stump.json_model.SendKey attribute*), 37
params (*xbox.stump.json_model.StumpRequest attribute*), 37
params (*xbox.stump.json_model.TunerLineups attribute*), 37
PKCS7Padding (*class in xbox.sg.crypto*), 24
playback_status() (*xbox.sg.manager.MediaManager property*), 9
position() (*xbox.sg.manager.MediaManager property*), 9
power_off() (*in module xbox.sg.factory*), 27
power_off() (*xbox.sg.console.Console method*), 4
power_off() (*xbox.sg.protocol.SmartglassProtocol method*), 17
power_on() (*in module xbox.sg.factory*), 25
power_on() (*xbox.sg.console.Console class method*), 3
power_on() (*xbox.sg.protocol.SmartglassProtocol method*), 16
Prepend (*xbox.sg.crypto.SaltType attribute*), 21
PROGRAM_CHANGED (*xbox.stump.enum.Notification attribute*), 36
PROGRAMM_INFO (*xbox.stump.enum.Message attribute*), 35
ProtocolError, 13
pubkey_bytes() (*xbox.sg.crypto.Crypto property*), 22
pubkey_type() (*xbox.sg.crypto.Crypto property*), 22
public_key() (*xbox.sg.console.Console property*), 5

Q

Quality (*class in xbox.stump.enum*), 36

R

rate() (*xbox.sg.manager.MediaManager property*), 9
reassemble_json() (*xbox.sg.protocol.FragmentManager method*), 19
reassemble_message() (*xbox.sg.protocol.FragmentManager method*), 19
RECENT_CHANNELS (*xbox.stump.enum.Message attribute*), 35
RecentChannels (*class in xbox.stump.json_model*), 37
remove() (*xbox.sg.crypto.Padding static method*), 24
request (*xbox.stump.json_model.StumpRequest attribute*), 37
request_app_channel_data() (*xbox.stump.manager.StumpManager method*), 40
request_app_channel_lineups() (*xbox.stump.manager.StumpManager method*), 40
request_app_channel_program_data() (*xbox.stump.manager.StumpManager method*), 40
request_ensure_stump_streaming_started() (*xbox.stump.manager.StumpManager method*), 41
request_headend_info() (*xbox.stump.manager.StumpManager method*), 39
request_live_tv_info() (*xbox.stump.manager.StumpManager method*), 40
request_program_info() (*xbox.stump.manager.StumpManager method*), 40
request_recent_channels() (*xbox.stump.manager.StumpManager method*), 41
request_stump_configuration() (*xbox.stump.manager.StumpManager method*), 39
request_tuner_lineups() (*xbox.stump.manager.StumpManager method*), 40

reset() (*xbox.sg.protocol.ChannelManager method*), 19
reset_session() (*xbox.sg.manager.TextManager method*), 11
response (*xbox.stump.json_model.StumpResponse attribute*), 37
run() (*xbox.auxiliary.relay.AuxiliaryRelayService method*), 34

S

Salt (*class in xbox.sg.crypto*), 21
SaltType (*class in xbox.sg.crypto*), 21
send() (*xbox.auxiliary.relay.ConsoleConnection method*), 33
SEND_KEY (*xbox.stump.enum.Message attribute*), 35
send_message() (*xbox.sg.console.Console method*), 3
send_message() (*xbox.sg.protocol.SmartglassProtocol method*), 13
send_stump_key() (*xbox.stump.manager.StumpManager method*), 41
send_systemtext_ack() (*xbox.sg.manager.TextManager method*), 11
send_systemtext_done() (*xbox.sg.manager.TextManager method*), 12
send_systemtext_input() (*xbox.sg.manager.TextManager method*), 11
SendKey (*class in xbox.stump.json_model*), 37
SequenceManager (*class in xbox.sg.protocol*), 17
SET_CHANNEL (*xbox.stump.enum.Message attribute*), 35
set_stump_channel_by_id() (*xbox.stump.manager.StumpManager method*), 40
set_stump_channel_by_name() (*xbox.stump.manager.StumpManager method*), 40
SET_TOP_BOX (*xbox.stump.enum.DeviceType attribute*), 36
shared_secret() (*xbox.sg.crypto.Crypto property*), 22
size() (*xbox.sg.crypto.Padding static method*), 23
SmartglassProtocol (*class in xbox.sg.protocol*), 13
sound_level() (*xbox.sg.manager.MediaManager property*), 9
Source (*class in xbox.stump.enum*), 36
SourceHttpQuery (*class in xbox.stump.enum*), 36
start() (*xbox.auxiliary.relay.ConsoleConnection method*), 33
start_channel() (*in module xbox.sg.factory*), 27
start_channel() (*xbox.sg.protocol.SmartglassProtocol method*), 16
start_title_channel() (*xbox.auxiliary.manager.TitleManager method*), 32
stop() (*xbox.auxiliary.relay.ConsoleConnection method*), 33
stop() (*xbox.sg.protocol.SmartglassProtocol method*), 13
stop_channel() (*in module xbox.sg.factory*), 27
STREAMING_ERROR (*xbox.stump.enum.Notification attribute*), 35
streaming_port() (*xbox.stump.manager.StumpManager property*), 41
StumpError (*class in xbox.stump.json_model*), 37
StumpException, 38
StumpJsonError, 37
StumpManager (*class in xbox.stump.manager*), 38
StumpNotification (*class in xbox.stump.json_model*), 37
StumpRequest (*class in xbox.stump.json_model*), 37
StumpResponse (*class in xbox.stump.json_model*), 37
systemtext_ack() (*in module xbox.sg.factory*), 29
systemtext_done() (*in module xbox.sg.factory*), 29
systemtext_input() (*in module xbox.sg.factory*), 28

T

text_input_scope() (*xbox.sg.manager.TextManager property*), 11
text_locale() (*xbox.sg.manager.TextManager property*), 11
text_options() (*xbox.sg.manager.TextManager property*), 11
text_prompt() (*xbox.sg.manager.TextManager property*), 11
text_session_id() (*xbox.sg.manager.TextManager property*), 11
TextManager (*class in xbox.sg.manager*), 10
TextManagerError, 10
title_auxiliary_stream() (*in module xbox.sg.factory*), 29
title_id() (*xbox.sg.manager.MediaManager property*), 9
title_launch() (*in module xbox.sg.factory*), 27
TitleManager (*class in xbox.auxiliary.manager*), 32
TitleManagerError, 32
to_dict() (*xbox.sg.console.Console method*), 2
TUNER (*xbox.stump.enum.DeviceType attribute*), 36
TUNER (*xbox.stump.enum.Source attribute*), 36
TUNER (*xbox.stump.enum.SourceHttpQuery attribute*), 36

TUNER_LINEUPS (*xbox.stump.enum.Message attribute*), 35
TunerLineups (*class in xbox.stump.json_model*), 37
TUNERSTATE_CHANGED (*xbox.stump.enum.Notification attribute*), 36
TV (*xbox.stump.enum.DeviceType attribute*), 36

U

unpack() (*in module xbox.auxiliary.packer*), 33
unsnap() (*in module xbox.sg.factory*), 28

V

verify() (*xbox.auxiliary.crypto.AuxiliaryStreamCrypto method*), 32
verify() (*xbox.sg.crypto.Crypto method*), 23
VIDEOFORMAT_CHANGED (*xbox.stump.enum.Notification attribute*), 36

W

wait() (*xbox.sg.console.Console static method*), 2

X

xbox.auxiliary
    module, 34
xbox.auxiliary.crypto
    module, 31
xbox.auxiliary.manager
    module, 32
xbox.auxiliary.packer
    module, 33
xbox.auxiliary.packet
    module, 33
xbox.auxiliary.relay
    module, 33
xbox.sg.console
    module, 1
xbox.sg.crypto
    module, 21
xbox.sg.factory
    module, 25
xbox.sg.manager
    module, 7
xbox.sg.protocol
    module, 13
xbox.stump
    module, 41
xbox.stump.enum
    module, 35
xbox.stump.json_model
    module, 37
xbox.stump.manager
    module, 38