adb Documentation

Release 1.3.0

Fahrzin Hemmati

Oct 18, 2019

CONTENTS

CHAPTER

ONE

ADB

1.1 adb package

1.1.1 Submodules

adb.adb_commands module

A libusb1-based ADB reimplementation.

ADB was giving us trouble with its client/server architecture, which is great for users and developers, but not so great for reliable scripting. This will allow us to more easily catch errors as Python exceptions instead of checking random exit codes, and all the other great benefits from not going through subprocess and a network socket.

All timeouts are in milliseconds.

Contents

- *AdbCommands.Reboot()*
- *AdbCommands.RebootBootloader()*
- *AdbCommands.Remount()*
- *AdbCommands.Root()*
- *AdbCommands.Shell()*
- *AdbCommands.Stat()*
- *AdbCommands.StreamingShell()*
- *AdbCommands.Uninstall()*

class adb.adb_commands.**AdbCommands**

Bases: `object`

Exposes adb-like methods for use.

Some methods are more-pythonic and/or have more options.

build_props

TODO

Type TODO, None

filesync_handler

TODO

Type *filesync_protocol.FilesyncProtocol*

protocol_handler

TODO

Type *adb_protocol.AdbMessage*

_device_state

TODO

Type TODO, None

_handle

TODO

Type *adb.common.TcpHandle*, *adb.common.UsbHandle*, None

_service_connections

[TODO] Connection table tracks each open AdbConnection objects per service type for program functions that choose to persist an AdbConnection object for their functionality, using *AdbCommands._get_service_connection()*

Type dict

Close()

TODO

ConnectDevice(*port_path=None*, *serial=None*, *default_timeout_ms=None*, ***kwargs*)

Convenience function to setup a transport handle for the adb device from usb path or serial then connect to it.

Parameters

- **`port_path`** (*`TODO, None`*) – The filename of usb port to use.
- **`serial`** (*`TODO, None`*) – The serial number of the device to use. If serial specifies a TCP address:port, then a TCP connection is used instead of a USB connection.
- **`default_timeout_ms`** (*`TODO, None`*) – The default timeout in milliseconds to use.
- **`**kwargs`** – Keyword arguments
- **`handle`** (`common.TcpHandle, common.UsbHandle`) – Device handle to use
- **`banner`** (*`TODO`*) – Connection banner to pass to the remote device
- **`rsa_keys`** (*`list[adb_protocol.AuthSigner]`*) – List of AuthSigner subclass instances to be used for authentication. The device can either accept one of these via the `Sign` method, or we will send the result of `GetPublicKey` from the first one if the device doesn't accept any of them.
- **`auth_timeout_ms`** (*`int`*) – Timeout to wait for when sending a new public key. This is only relevant when we send a new public key. The device shows a dialog and this timeout is how long to wait for that dialog. If used in automation, this should be low to catch such a case as a failure quickly; while in interactive settings it should be high to allow users to accept the dialog. We default to automation here, so it's low by default.

Returns **self** – TODO

Return type *AdbCommands*

`classmethod Devices()`

Get a generator of *`UsbHandle`* for devices available.

Returns TODO

Return type TODO

`DisableVerity()`

Disable dm-verity checking on userdebug builds.

Returns TODO

Return type TODO

`EnableVerity()`

Re-enable dm-verity checking on userdebug builds.

Returns TODO

Return type TODO

`GetState()`

TODO

Returns **self._device_state** – TODO

Return type TODO

`Install`(*apk_path*, *destination_dir=”*, *replace_existing=True*, *grant_permissions=False*, *timeout_ms=None*, *transfer_progress_callback=None*)

Install an apk to the device.

Doesn't support verifier file, instead allows destination directory to be overridden.

Parameters

- **`apk_path`** (*`TODO`*) – Local path to apk to install.
- **`destination_dir`** (*`str`*) – Optional destination directory. Use `/system/app/` for persistent applications.
- **`replace_existing`** (*`bool`*) – Whether to replace existing application
- **`grant_permissions`** (*`bool`*) – If `True`, grant all permissions to the app specified in its manifest
- **`timeout_ms`** (*`int, None`*) – Expected timeout for pushing and installing.
- **`transfer_progress_callback`** (*`TODO, None`*) – callback method that accepts `filename`, `bytes_written`, and `total_bytes` of APK transfer

Returns **ret** – The `pm install` output.

Return type TODO

`InteractiveShell`(*cmd=None*, *strip_cmd=True*, *delim=None*, *strip_delim=True*)

Get stdout from the currently open interactive shell and optionally run a command on the device, returning all output.

Parameters

- **`cmd`** (*`TODO, None`*) – Command to run on the target.
- **`strip_cmd`** (*`bool`*) – Strip command name from stdout.
- **`delim`** (*`TODO, None`*) – Delimiter to look for in the output to know when to stop expecting more output (usually the shell prompt)
- **`strip_delim`** (*`bool`*) – Strip the provided delimiter from the output

Returns The stdout from the shell command.

Return type TODO

`List`(*device_path*)

Return a directory listing of the given path.

Parameters **`device_path`** (*`TODO`*) – Directory to list.

Returns **listing** – TODO

Return type TODO

`Logcat`(*options*, *timeout_ms=None*)

Run `shell logcat` and stream the output to stdout.

Parameters

- **`options`** (*`str`*) – Arguments to pass to `logcat`
- **`timeout_ms`** (*`int, None`*) – Maximum time to allow the command to run.

Returns The responses from the `logcat` command

Return type generator

`Pull`(*device_filename*, *dest_file=None*, *timeout_ms=None*, *progress_callback=None*)

Pull a file from the device.

Parameters

- **`device_filename`** (*`TODO`*) – Filename on the device to pull.

- **dest_file** (*str, file, io.IOBase, None*) – If set, a filename or writable file-like object.
- **timeout_ms** (*int, None*) – Expected timeout for any part of the pull.
- **progress_callback** (*TODO, None*) – Callback method that accepts filename, bytes_written and total_bytes, total_bytes will be -1 for file-like objects

Returns The file data if `dest_file` is not set. Otherwise, `True` if the destination file exists

Return type TODO

Raises **ValueError** – If `dest_file` is of unknown type.

Push (*source_file*, *device_filename*, *mtime='0'*, *timeout_ms=None*, *progress_callback=None*, *st_mode=None*)
Push a file or directory to the device.

Parameters

- **source_file** (*TODO*) – Either a filename, a directory or file-like object to push to the device.
- **device_filename** (*TODO*) – Destination on the device to write to.
- **mtime** (*str*) – Modification time to set on the file.
- **timeout_ms** (*int, None*) – Expected timeout for any part of the push.
- **progress_callback** (*TODO, None*) – Callback method that accepts filename, bytes_written and total_bytes, total_bytes will be -1 for file-like objects
- **st_mode** (*TODO, None*) – Stat mode for filename

Reboot (*destination=b''*)
Reboot the device.

Parameters **destination** (*bytes*) – Specify `'bootloader'` for fastboot.

RebootBootloader ()
Reboot device into fastboot.

Remount ()
Remount / as read-write.

Returns TODO

Return type TODO

Root ()
Restart `adbd` as root on the device.

Returns TODO

Return type TODO

Shell (*command*, *timeout_ms=None*)
Run command on the device, returning the output.

Parameters

- **command** (*TODO*) – Shell command to run
- **timeout_ms** (*int, None*) – Maximum time to allow the command to run.

Returns TODO

Return type TODO

`Stat`(*device_filename*)
Get a file's `stat()` information.

Parameters **`device_filename`** (*`TODO`*) – TODO

Returns

- **mode** (*TODO*) – TODO
- **size** (*TODO*) – TODO
- **mtime** (*TODO*) – TODO

`StreamingShell`(*command*, *timeout_ms=None*)
Run command on the device, yielding each line of output.

Parameters

- **`command`** (*`bytes`*) – Command to run on the target.
- **`timeout_ms`** (*`int, None`*) – Maximum time to allow the command to run.

Returns The responses from the shell command.

Return type generator

`Uninstall`(*package_name*, *keep_data=False*, *timeout_ms=None*)
Removes a package from the device.

Parameters

- **`package_name`** (*`TODO`*) – Package name of target package.
- **`keep_data`** (*`bool`*) – Whether to keep the data and cache directories
- **`timeout_ms`** (*`int, None`*) – Expected timeout for pushing and installing.

Returns The `pm uninstall` output.

Return type TODO

`_AdbCommands__reset`()
TODO

`_Connect`(*banner=None*, ***kwargs*)
Connect to the device.

Parameters

- **`banner`** (*`bytes, None`*) – A string to send as a host identifier. (See *`adb.adb_protocol.AdbMessage.Connect()`*.)
- **`**kwargs`** (*`TODO`*) – See *`adb.adb_protocol.AdbMessage.Connect()`* and *`AdbCommands.ConnectDevice()`* for kwargs. Includes `handle`, `rsa_keys`, and `auth_timeout_ms`.

Returns `True`

Return type bool

`_get_service_connection`(*service*, *service_command=None*, *create=True*, *timeout_ms=None*)
Based on the service, get the AdbConnection for that service or create one if it doesnt exist

Parameters

- **`service`** (*`TODO`*) – TODO
- **`service_command`** (*`TODO, None`*) – Additional service parameters to append
- **`create`** (*`bool`*) – If False, don't create a connection if it does not exist
- **`timeout_ms`** (*`int, None`*) – TODO

Returns **connection** – TODO

Return type TODO

`filesync_handler`
alias of *`adb.filesync_protocol.FilesyncProtocol`*

`protocol_handler`
alias of *`adb.adb_protocol.AdbMessage`*

`adb.adb_commands.`**`CLASS = 255`**
From adb.h

`adb.adb_commands.`**`DeviceIsAvailable`**(*device*)
From adb.h

`adb.adb_commands.`**`PROTOCOL = 1`**
From adb.h

`adb.adb_commands.`**`SUBCLASS = 66`**
From adb.h

adb.adb_debug module

Daemon-less ADB client in python.

Contents

- *`Devices()`*
- *`List()`*
- `Logcat()`
- *`Shell()`*

`adb.adb_debug.`**`Devices`**(*args*)
Lists the available devices.

Mimics `adb devices` output:

```
List of devices attached
015DB7591102001A        device        1,2
```

See also:

`adb.adb_commands.AdbCommands.Devices()`

Parameters **`args`** (*`argparse.ArgumentParser`*) – CLI arguments; see *`adb.common_cli.GetDeviceArguments()`*.

Returns 0

Return type int

adb.adb_debug.**List** (*device*, *device_path*)

Prints a directory listing.

See also:

adb.adb_commands.AdbCommands.List()

Parameters

- **device** (adb.adb_commands.AdbCommands) – TODO
- **device_path** (*str, bytes*) – Directory to list.

Yields *str* – A formatted listing for a file in `device_path`

adb.adb_debug.**Shell** (*device*, **command*)

Runs a command on the device and prints the stdout.

See also:

adb.adb_commands.AdbCommands.StreamingShell(), *adb.adb_commands.AdbCommands.InteractiveShell()*

Parameters

- **device** (adb.adb_commands.AdbCommands) – TODO
- **command** (*list[str]*) – Command to run on the target.

adb.adb_debug.**main** ()

TODO

adb.adb_keygen module

This file implements encoding and decoding logic for Android's custom RSA public key binary format. Public keys are stored as a sequence of little-endian 32 bit words. Note that Android only supports little-endian processors, so we don't do any byte order conversions when parsing the binary struct.

Structure from: https://github.com/aosp-mirror/platform_system_core/blob/c55fab4a59cfa461857c6a61d8a0f1ae4591900c/libcrypto_utils/android_pubkey.c

```
typedef struct RSAPublicKey {
    // Modulus length. This must be ANDROID_PUBKEY_MODULUS_SIZE_WORDS
    uint32_t modulus_size_words;

    // Precomputed montgomery parameter: -1 / n[0] mod 2^32
    uint32_t n0inv;

    // RSA modulus as a little-endian array
    uint8_t modulus[ANDROID_PUBKEY_MODULUS_SIZE];

    // Montgomery parameter R^2 as a little-endian array of little-endian words
    uint8_t rr[ANDROID_PUBKEY_MODULUS_SIZE];

    // RSA modulus: 3 or 65537
```

(continues on next page)

(continued from previous page)

```
    uint32_t exponent;
} RSAPublicKey;
```

Contents

adb.adb_keygen.**ANDROID_PUBKEY_MODULUS_SIZE = 256**
Size of an RSA modulus such as an encrypted block or a signature.

adb.adb_keygen.**ANDROID_PUBKEY_MODULUS_SIZE_WORDS = 64**
Size of the RSA modulus in words.

adb.adb_keygen.**ANDROID_RSAPUBLICKEY_STRUCT = '<LL256s256sL'**
Python representation of "struct RSAPublicKey"

adb.adb_keygen.**_to_bytes**(*n*, *length*, *endianess='big'*)
Partial python2 compatibility with int.to_bytes

https://stackoverflow.com/a/20793663

Parameters

- **n** (*TODO*) – TODO
- **length** (*TODO*) – TODO
- **endianess** (*str, TODO*) – TODO

Returns TODO

Return type TODO

adb.adb_keygen.**decode_pubkey**(*public_key*)
Decode a public RSA key stored in Android's custom binary format.

Parameters **public_key** (*TODO*) – TODO

adb.adb_keygen.**decode_pubkey_file**(*public_key_path*)
TODO

Parameters **public_key_path** (*str*) – TODO

adb.adb_keygen.**encode_pubkey**(*private_key_path*)
Encodes a public RSA key into Android's custom binary format.

Parameters **private_key_path** (*str*) – TODO

Returns TODO

Return type TODO

`adb.adb_keygen.`**`get_user_info`**`()`

TODO

Returns `' <username>@<hostname>`

Return type str

`adb.adb_keygen.`**`keygen`**`(`*filepath*`)`

Generate an ADB public/private key pair.

- The private key is stored in `filepath`.
- The public key is stored in `filepath + '.pub'`

(Existing files will be overwritten.)

Parameters **`filepath`** (*`str`*) – File path to write the private/public keypair

`adb.adb_keygen.`**`write_public_keyfile`**`(`*private_key_path*, *public_key_path*`)`

Write a public keyfile to `public_key_path` in Android's custom RSA public key format given a path to a private keyfile.

Parameters

- **`private_key_path`** (*`TODO`*) – TODO
- **`public_key_path`** (*`TODO`*) – TODO

adb.adb_protocol module

ADB protocol implementation.

Implements the ADB protocol as seen in android's adb/adbd binaries, but only the host side.

Contents

adb.adb_protocol.**AUTH_RSAPUBLICKEY = 3**
AUTH constants for arg0.

adb.adb_protocol.**AUTH_SIGNATURE = 2**
AUTH constants for arg0.

adb.adb_protocol.**AUTH_TOKEN = 1**
AUTH constants for arg0.

class adb.adb_protocol.**AdbMessage**(*command=None*, *arg0=None*, *arg1=None*, *data=b''*)
Bases: object

ADB Protocol and message class.

local_id/remote_id

Turns out the documentation is host/device ambidextrous, so local_id is the id for 'the sender' and remote_id is for 'the recipient'. So since we're only on the host, we'll re-document with host_id and device_id:

```
OPEN(host_id, 0, 'shell:XXX')
READY/OKAY(device_id, host_id, '')
WRITE(0, host_id, 'data')
CLOSE(device_id, host_id, '')
```

Parameters

- **command** (*bytes, None*) – One of: [b'SYNC', b'CNXN', b'AUTH', b'OPEN', b'OKAY', b'CLSE', b'WRTE']
- **arg0** (*TODO, None*) – TODO
- **arg1** (*TODO, None*) – TODO
- **data** (*bytes*) – TODO

command
The value in *AdbMessage.commands* that corresponds to the command parameter

Type int

`commands`
A dictionary with keys `[b'SYNC', b'CNXN', b'AUTH', b'OPEN', b'OKAY', b'CLSE', b'WRTE']`.

Type dict

`connections`
TODO

Type int

`constants`
A dictionary with values `[b'SYNC', b'CNXN', b'AUTH', b'OPEN', b'OKAY', b'CLSE', b'WRTE']`.

Type dict

`format`
The format for unpacking the ADB message.

Type bytes

`ids`
`[b'SYNC', b'CNXN', b'AUTH', b'OPEN', b'OKAY', b'CLSE', b'WRTE']`

Type list[bytes]

`static CalculateChecksum`(*data*)
TODO

Returns TODO

Return type TODO

`classmethod Command`(*usb*, *service*, *command=''*, *timeout_ms=None*)
One complete set of USB packets for a single command.

Sends `service:command` in a new connection, reading the data for the response. All the data is held in memory, large responses will be slow and can fill up memory.

Parameters

- **usb** (`adb.common.TcpHandle`, `adb.common.UsbHandle`) – A *`adb.common.TcpHandle`* or *`adb.common.UsbHandle`* instance with `BulkRead` and `BulkWrite` methods.
- **service** (*TODO*) – The service on the device to talk to.
- **command** (*str*) – The command to send to the service.
- **timeout_ms** (*int, None*) – Timeout in milliseconds for USB packets.

Returns The response from the service.

Return type str

Raises

- ***`adb.adb_protocol.InterleavedDataError`*** – Multiple streams running over usb.
- ***`adb.adb_protocol.InvalidCommandError`*** – Got an unexpected response command.

classmethod Connect (*usb*, *banner=b'notadb'*, *rsa_keys=None*, *auth_timeout_ms=100*)

Establish a new connection to the device.

Parameters

- **usb** (`adb.common.TcpHandle`, `adb.common.UsbHandle`) – A *adb.common.TcpHandle* or *adb.common.UsbHandle* instance with `BulkRead` and `BulkWrite` methods.
- **banner** (*str*) – A string to send as a host identifier.
- **rsa_keys** (*list[*adb_protocol.AuthSigner*]*) – List of *AuthSigner* subclass instances to be used for authentication. The device can either accept one of these via the `Sign` method, or we will send the result of `GetPublicKey` from the first one if the device doesn't accept any of them.
- **auth_timeout_ms** (*int*) – Timeout to wait for when sending a new public key. This is only relevant when we send a new public key. The device shows a dialog and this timeout is how long to wait for that dialog. If used in automation, this should be low to catch such a case as a failure quickly; while in interactive settings it should be high to allow users to accept the dialog. We default to automation here, so it's low by default.

Returns **banner** – The device's reported banner. Always starts with the state (device, recovery, or sideload), sometimes includes information after a : with various product information.

Return type TODO

Raises

- ***adb.usb_exceptions.DeviceAuthError*** – When the device expects authentication, but we weren't given any valid keys.
- ***adb.adb_protocol.InvalidResponseError*** – When the device does authentication in an unexpected way.
- ***usb_exceptions.ReadFailedError*** – TODO

classmethod InteractiveShellCommand (*conn*, *cmd=None*, *strip_cmd=True*, *delim=None*, *strip_delim=True*, *clean_stdout=True*)

Retrieves stdout of the current InteractiveShell and sends a shell command if provided TODO: Should we turn this into a yield based function so we can stream all output?

Parameters

- **conn** (*AdbConnection*) – Instance of AdbConnection
- **cmd** (*str, None*) – Command to run on the target.
- **strip_cmd** (*bool*) – Strip command name from stdout.
- **delim** (*TODO*) – Delimiter to look for in the output to know when to stop expecting more output (usually the shell prompt)
- **strip_delim** (*bool*) – Strip the provided delimiter from the output
- **clean_stdout** (*bool*) – Cleanup the stdout stream of any backspaces and the characters that were deleted by the backspace

Returns **stdout** – The stdout from the shell command.

Return type TODO

classmethod Open (*usb*, *destination*, *timeout_ms=None*)

Opens a new connection to the device via an `OPEN` message.

Not the same as the posix `open` or any other google3 Open methods.

Parameters

- **`usb`** (`adb.common.TcpHandle`, `adb.common.UsbHandle`) – A *`adb.common.TcpHandle`* or *`adb.common.UsbHandle`* instance with `BulkRead` and `BulkWrite` methods.
- **`destination`** (*`TODO`*) – The service:command string.
- **`timeout_ms`** (*`int`*, *`None`*) – Timeout in milliseconds for USB packets.

Returns The local connection id.

Return type *_AdbConnection*, None

Raises

- ***`adb.adb_protocol.InvalidResponseError`*** – Wrong local_id sent to us.
- ***`adb.adb_protocol.InvalidCommandError`*** – Didn't get a ready response.

`Pack`()

Returns this message in an over-the-wire format.

Returns TODO

Return type bytes

`classmethod Read`(*usb*, *expected_cmds*, *timeout_ms=None*, *total_timeout_ms=None*)

Receive a response from the device.

Parameters

- **`usb`** (`adb.common.TcpHandle`, `adb.common.UsbHandle`) – TODO
- **`expected_cmds`** (*`TODO`*) – Read until we receive a header ID that is in `expected_cmds`
- **`timeout_ms`** (*`int`*, *`None`*) – Timeout in milliseconds for USB packets.
- **`total_timeout_ms`** (*`int`*, *`None`*) – The total time to wait for a command in `expected_cmds`

Returns

- **command** (*TODO*) – TODO
- **arg0** (*TODO*) – TODO
- **arg1** (*TODO*) – TODO
- *bytes* – TODO

Raises

- ***`adb.adb_protocol.InvalidCommandError`*** – Unknown command *or* never got one of the expected responses.
- ***`adb.adb_protocol.InvalidChecksumError`*** – Received checksum does not match the expected checksum.

`Send`(*usb*, *timeout_ms=None*)

Send this message over USB.

Parameters

- **`usb`** (`adb.common.TcpHandle, adb.common.UsbHandle`) – TODO
- **`timeout_ms`** (*`int, None`*) – Timeout in milliseconds for USB packets.

`classmethod StreamingCommand`(*usb*, *service*, *command=''*, *timeout_ms=None*)

One complete set of USB packets for a single command.

Sends `service:command` in a new connection, reading the data for the response. All the data is held in memory, large responses will be slow and can fill up memory.

Parameters

- **`usb`** (`adb.common.TcpHandle, adb.common.UsbHandle`) – A *`adb.common.TcpHandle`* or *`adb.common.UsbHandle`* instance with `BulkRead` and `BulkWrite` methods.
- **`service`** (*`TODO`*) – The service on the device to talk to.
- **`command`** (*`str`*) – The command to send to the service.
- **`timeout_ms`** (*`int, None`*) – Timeout in milliseconds for USB packets.

Yields *str* – The responses from the service.

Raises

- ***`adb.adb_protocol.InterleavedDataError`*** – Multiple streams running over usb.
- ***`adb.adb_protocol.InvalidCommandError`*** – Got an unexpected response command.

`classmethod Unpack`(*message*)

TODO

Parameters **`message`** (*`TODO`*) – TODO

Returns

- **cmd** (*TODO*) – TODO
- **arg0** (*TODO*) – TODO
- **arg1** (*TODO*) – TODO
- **data_length** (*TODO*) – TODO
- **data_checksum** (*TODO*) – TODO
- **unused_magic** (*TODO*) – TODO

Raises **`ValueError`** – Unable to unpack the ADB command.

`property checksum`

TODO

Returns TODO

Return type TODO

`commands = {b'AUTH': 1213486401, b'CLSE': 1163086915, b'CNXN': 1314410051, b'OKAY': 14`

`connections = 0`

constants = {1129208147: b'SYNC', 1163086915: b'CLSE', 1163154007: b'WRTE', 1213486

format = b'<6I'

ids = [b'SYNC', b'CNXN', b'AUTH', b'OPEN', b'OKAY', b'CLSE', b'WRTE']

class `adb.adb_protocol.`**AuthSigner**

Bases: `object`

Signer for use with authenticated ADB, introduced in 4.4.x/KitKat.

GetPublicKey()

Returns the public key in PEM format without headers or newlines.

Raises **NotImplementedError** – This method is implemented in subclasses.

Sign(*data*)

Signs given data using a private key.

Parameters **data** (*bytes*) – The data to be signed

Raises **NotImplementedError** – This method is implemented in subclasses.

exception `adb.adb_protocol.`**InterleavedDataError**

Bases: `Exception`

We only support command sent serially.

exception `adb.adb_protocol.`**InvalidChecksumError**

Bases: `Exception`

Checksum of data didn't match expected checksum.

exception `adb.adb_protocol.`**InvalidCommandError**(*message*, *response_header*, *response_data*)

Bases: `Exception`

Got an invalid command over USB.

exception `adb.adb_protocol.`**InvalidResponseError**

Bases: `Exception`

Got an invalid response to our command.

`adb.adb_protocol.`**MAX_ADB_DATA = 4096**

Maximum amount of data in an ADB packet.

`adb.adb_protocol.`**MakeWireIDs**(*ids*)

TODO

Parameters **ids** (*list[bytes]*) – TODO

Returns

- **id_to_wire** (*dict*) – TODO
- **wire_to_id** (*dict*) – TODO

adb.adb_protocol.**VERSION = 16777216**

ADB protocol version.

class adb.adb_protocol.**_AdbConnection**(*usb*, *local_id*, *remote_id*, *timeout_ms*)

Bases: object

ADB Connection.

Parameters

- **usb** (adb.common.UsbHandle) – TODO
- **local_id** (*TODO*) – TODO
- **remote_id** (*TODO*) – TODO
- **timeout_ms** (*int*) – Timeout in milliseconds for USB packets.

local_id

The ID for the sender

Type TODO

remote_id

The ID for the recipient

Type TODO

timeout_ms

Timeout in milliseconds for USB packets.

Type int

usb

TODO

Type *adb.common.UsbHandle*

Close()

TODO

Raises

- ***usb_exceptions.AdbCommandFailureException*** – Command failed.
- ***adb.adb_protocol.InvalidCommandError*** – Expected a CLSE response but received something else.

Okay()

TODO

ReadUntil(**expected_cmds*)

Read a packet, Ack any write packets.

Parameters ***expected_cmds** (*TODO*) – TODO

Returns

- **cmd** (*TODO*) – TODO

- **data** (*TODO*) – TODO

Raises

- ***adb.adb_protocol.InterleavedDataError*** – We don't support multiple streams. . .
- ***adb.adb_protocol.InvalidResponseError*** – Incorrect remote id.

`ReadUntilClose``()`
Yield packets until a `b'CLSE'` packet is received.

Yields **data** (*TODO*) – TODO

`Write``(`*data*`)`
Write a packet and expect an Ack.

Parameters **`data`** (*TODO*) – TODO

Returns `len(data)`

Return type int

Raises

- ***usb_exceptions.AdbCommandFailureException*** – The command failed.
- ***adb.adb_protocol.InvalidCommandError*** – Expected an OKAY in response to a WRITE, got something else.

`_Send``(`*command*, *arg0*, *arg1*, *data=b''*`)`
TODO

Parameters

- **`command`** (*TODO*) – TODO
- **`arg0`** (*TODO*) – TODO
- **`arg1`** (*TODO*) – TODO
- **`data`** (*bytes*) – TODO

`adb.adb_protocol.`**`find_backspace_runs`**`(`*stdout_bytes*, *start_pos*`)`
TODO

Parameters

- **`stdout_bytes`** (*TODO*) – TODO
- **`start_pos`** (*TODO*) – TODO

Returns

- *int* – The index/position of the first backspace.
- **num_backspaces** (*int*) – TODO

adb.common module

Common code for ADB and Fastboot.

Common usb browsing and usb communication.

Contents

`adb.common.`**`DEFAULT_TIMEOUT_MS = 10000`**
Default timeout

`adb.common.`**`GetInterface`**`(`*setting*`)`
Get the class, subclass, and protocol for the given USB setting.

Parameters **`setting`** (*TODO*) – TODO

Returns

- *TODO* – TODO
- *TODO* – TODO

- *TODO* – TODO

`adb.common.`**`InterfaceMatcher`**(*clazz*, *subclass*, *protocol*)

Returns a matcher that returns the setting with the given interface.

Parameters

- **`clazz`** (*`TODO`*) – TODO
- **`subclass`** (*`TODO`*) – TODO
- **`protocol`** (*`TODO`*) – TODO

Returns **Matcher** – TODO

Return type function

`class` `adb.common.`**`TcpHandle`**(*serial*, *timeout_ms=None*)

Bases: `object`

TCP connection object.

Provides same interface as *UsbHandle*.

Parameters

- **`serial`** (*`str, bytes, bytearray`*) – Android device serial of the form "host" or "host:port". (Host may be an IP address or a host name.)
- **`timeout_ms`** (*`TODO, None`*) – TODO

`_connection`

TODO

Type TODO, None

`_serial_number`

`<host>:<port>`

Type str

`_timeout_ms`

TODO

Type float, None

`host`

TODO

Type str, TODO

`port`

TODO

Type str, int, TODO

`BulkRead`(*numbytes*, *timeout=None*)

TODO

Parameters

- **`numbytes`** (*`int`*) – TODO
- **`timeout_ms`** (*`TODO, None`*) – TODO

Returns TODO

Return type TODO

Raises `adb.usb_exceptions.TcpTimeoutException` – Reading timed out.

`BulkWrite`(*data*, *timeout=None*)
TODO

Parameters

- **`data`** (*TODO*) – TODO
- **`timeout`** (*TODO, None*) – TODO

Returns TODO

Return type TODO

Raises `adb.usb_exceptions.TcpTimeoutException` – Sending data timed out. No data was sent.

`Close`()
TODO

Returns TODO

Return type TODO

`Timeout`(*timeout_ms*)
TODO

Parameters **`timeout_ms`** (*TODO*) – TODO

Returns TODO

Return type float

`TimeoutSeconds`(*timeout_ms*)
TODO

Parameters **`timeout_ms`** (*TODO*) – TODO

Returns TODO

Return type float

`_connect`()
TODO

`property serial_number`
TODO

Returns **self._serial_number** – The `_serial_number` attribute (`<host>:<port>`)

Return type str

`class` `adb.common.`**`UsbHandle`**(*device*, *setting*, *usb_info=None*, *timeout_ms=None*)
Bases: `object`

USB communication object. Not thread-safe.

Handles reading and writing over USB with the proper endpoints, exceptions, and interface claiming.

Important methods: * *UsbHandle.FlushBuffers* * *UsbHandle.BulkRead* * *UsbHandle.BulkWrite(bytes data)*

Parameters

- **`device`** (*TODO*) – libusb_device to connect to.

- **setting** (*TODO*) – libusb setting with the correct endpoints to communicate with.
- **usb_info** (*TODO, None*) – String describing the usb path/serial/device, for debugging.
- **timeout_ms** (*TODO, None*) – Timeout in milliseconds for all I/O.

_device
libusb_device to connect to.

Type TODO

_handle
TODO

Type TODO

_interface_number
TODO

Type TODO

_max_read_packet_len
TODO

Type TODO

_read_endpoint
TODO

Type TODO

_setting
libusb setting with the correct endpoints to communicate with.

Type TODO

_timeout_ms
Timeout in milliseconds for all I/O.

Type TODO, None

_usb_info
String describing the usb path/serial/device, for debugging.

Type TODO

_write_endpoint
TODO

Type TODO, None

BulkRead (*length*, *timeout_ms=None*)
TODO

Parameters

- **length** (*int*) – TODO
- **timeout_ms** (*TODO, None*) – TODO

Returns TODO

Return type bytearray

Raises *usb_exceptions.ReadFailedError* – Could not receive data

BulkReadAsync(*length*, *timeout_ms=None*)
TODO

Parameters

- **length** (*int*) – TODO
- **timeout_ms** (*TODO, None*) – TODO

Raises **NotImplementedError** – This is always raised because this method is not implemented.

BulkWrite(*data*, *timeout_ms=None*)
TODO

Parameters

- **data** (*bytes*) – TODO
- **timeout_ms** (*TODO, None*) – TODO

Returns TODO

Return type TODO

Raises

- *adb.usb_exceptions.WriteFailedError* – This handle has been closed, probably due to another being opened
- *adb.usb_exceptions.WriteFailedError* – Could not send data

Close()
TODO

classmethod Find(*setting_matcher*, *port_path=None*, *serial=None*, *timeout_ms=None*)
Gets the first device that matches according to the keyword args.

Parameters

- **setting_matcher** (*TODO*) – TODO
- **port_path** (*TODO, None*) – TODO
- **serial** (*TODO, None*) – TODO
- **timeout_ms** (*TODO, None*) – TODO

Returns TODO

Return type TODO

classmethod FindAndOpen(*setting_matcher*, *port_path=None*, *serial=None*, *timeout_ms=None*)
TODO

Parameters

- **setting_matcher** (*TODO*) – TODO
- **port_path** (*TODO, None*) – TODO

- **serial** (*TODO, None*) – TODO
- **timeout_ms** (*TODO, None*) – TODO

Returns dev – TODO

Return type TODO

classmethod FindDevices (*setting_matcher, device_matcher=None, usb_info=”, timeout_ms=None*)

Find and yield the devices that match.

Parameters

- **setting_matcher** (*TODO*) – Function that returns the setting to use given a `usb1.USBDevice`, or `None` if the device doesn’t have a valid setting.
- **device_matcher** (*TODO, None*) – Function that returns `True` if the given `UsbHandle` is valid. `None` to match any device.
- **usb_info** (*str*) – Info string describing device(s).
- **timeout_ms** (*TODO, None*) – Default timeout of commands in milliseconds.

Yields *TODO* – UsbHandle instances

classmethod FindFirst (*setting_matcher, device_matcher=None, **kwargs*)

Find and return the first matching device.

Parameters

- **setting_matcher** (*TODO*) – See `UsbHandle.FindDevices()`.
- **device_matcher** (*TODO*) – See `UsbHandle.FindDevices()`.
- ****kwargs** (*TODO*) – See `UsbHandle.FindDevices()`.

Returns An instance of UsbHandle.

Return type TODO

Raises *adb.usb_exceptions.DeviceNotFoundError* – Raised if the device is not available.

FlushBuffers ()

TODO

Raises *adb.usb_exceptions.ReadFailedError* – TODO

Open ()

Opens the USB device for this setting, and claims the interface.

classmethod PortPathMatcher (*port_path*)

Returns a device matcher for the given port path.

Parameters **port_path** (*TODO*) – TODO

Returns TODO

Return type function

classmethod SerialMatcher (*serial*)

Returns a device matcher for the given serial.

Parameters **serial** (*TODO*) – TODO

Returns TODO

Return type function

Timeout (*timeout_ms*)
TODO

Returns TODO

Return type TODO

_HANDLE_CACHE = <WeakValueDictionary>

_HANDLE_CACHE_LOCK = <unlocked _thread.lock object>

property port_path
TODO

Returns TODO

Return type TODO

property serial_number
TODO

Returns TODO

Return type TODO

property usb_info
TODO

Returns TODO

Return type TODO

adb.common_cli module

Common code for ADB and Fastboot CLI.

Usage introspects the given class for methods, args, and docs to show the user.

`StartCli()` handles connecting to a device, calling the expected method, and outputting the results.

Contents

`adb.common_cli.`**`GetCommonArguments`**`()`

Return a parser for common CLI commands.

Returns **group** – A parser for common CLI commands.

Return type argparse.ArgumentParser

`adb.common_cli.`**`GetDeviceArguments`**`()`

Return a parser for device-related CLI commands.

Returns **group** – A parser for device-related CLI commands.

Return type argparse.ArgumentParser

`adb.common_cli.`**`MakeSubparser`**`(`*subparsers*, *parents*, *method*, *arguments=None*`)`

Returns an argparse subparser to create a 'subcommand' to adb.

Parameters

- **`subparsers`** (*`TODO`*) – TODO
- **`parents`** (*`TODO`*) – TODO
- **`method`** (*`TODO`*) – TODO
- **`arguments`** (*`TODO, None`*) – TODO

Returns **subparser** – TODO

Return type TODO

`class` `adb.common_cli.`**`PositionalArg`**`(`*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*`)`

Bases: `argparse.Action`

A positional CLI argument.

`adb.common_cli.`**`StartCli`**`(`*args*, *adb_commands*, *extra=None*, ***device_kwargs*`)`

Starts a common CLI interface for this usb path and protocol.

Handles connecting to a device, calling the expected method, and outputting the results.

Parameters

- **`args`** (*`TODO`*) – TODO
- **`adb_commands`** (*`TODO`*) – TODO
- **`extra`** (*`TODO, None`*) – TODO
- **`**device_kwargs`** (*`TODO`*) – TODO

Returns TODO

Return type TODO

`adb.common_cli.`**`_DocToArgs`**`(`*doc*`)`

Converts a docstring documenting arguments into a dict.

Parameters **`doc`** (*`str`*) – The docstring for a method; see *MakeSubparser*.

Returns **out** – A dictionary of arguments and their descriptions from the docstring `doc`.

Return type dict

class adb.common_cli.**_PortPathAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `argparse.Action`

TODO

adb.common_cli.**_RunMethod**(*dev*, *args*, *extra*)

Runs a method registered via `MakeSubparser()`.

Parameters

- **dev** (*TODO*) – TODO
- **args** (*TODO*) – TODO
- **extra** (*TODO*) – TODO

Returns 0

Return type int

adb.debug module

TEMPORARY.

adb.debug.**debug_print**(*name*, *var*)

Print debugging info.

adb.fastboot module

A libusb1-based fastboot implementation.

Contents

adb.fastboot.**CLASS = 255**
From fastboot.c

adb.fastboot.**DEFAULT_MESSAGE_CALLBACK**(*m*)
TODO

adb.fastboot.**DeviceIsAvailable**(*device*)
TODO

class adb.fastboot.**FastbootCommands**
Bases: `object`

Encapsulates the fastboot commands.

_handle
TODO

Type TODO, None

_protocol
TODO

Type TODO, None

Close()
TODO

ConnectDevice(*port_path=None*, *serial=None*, *default_timeout_ms=None*, *chunk_kb=1024*, ***kwargs*)
Convenience function to get an adb device from usb path or serial.

Parameters

- **port_path** (*TODO, None*) – The filename of usb port to use.
- **serial** (*TODO, None*) – The serial number of the device to use. If serial specifies a TCP address:port, then a TCP connection is used instead of a USB connection.
- **default_timeout_ms** (*TODO, None*) – The default timeout in milliseconds to use.

- **chunk_kb** (*int*) – Amount of data, in kilobytes, to break fastboot packets up into
- ****kwargs** (*dict*) – Keyword arguments
- **handle** (`adb.common.TcpHandle`, `adb.common.UsbHandle`) – Device handle to use
- **banner** (*TODO*) – Connection banner to pass to the remote device
- **rsa_keys** (*list[*`adb_protocol.AuthSigner`*]*) – List of AuthSigner subclass instances to be used for authentication. The device can either accept one of these via the `Sign` method, or we will send the result of `GetPublicKey` from the first one if the device doesn't accept any of them.
- **auth_timeout_ms** (*TODO*) – Timeout to wait for when sending a new public key. This is only relevant when we send a new public key. The device shows a dialog and this timeout is how long to wait for that dialog. If used in automation, this should be low to catch such a case as a failure quickly; while in interactive settings it should be high to allow users to accept the dialog. We default to automation here, so it's low by default.

Returns **self** – TODO

Return type *FastbootCommands*

Continue()

Continues execution past fastboot into the system.

Returns TODO

Return type TODO

classmethod Devices()

Get a generator of UsbHandle for devices available.

Returns TODO

Return type TODO

Download(*source_file*, *source_len=0*, *info_cb=<function <lambda>>*, *progress_callback=None*)

Downloads a file to the device.

Parameters

- **source_file** (*TODO*) – A filename or file-like object to download to the device.
- **source_len** (*int*) – Optional length of source_file. If `source_file` is a file-like object and `source_len` is not provided, `source_file` is read into memory.
- **info_cb** (*TODO*) – Optional callback accepting FastbootMessage for text sent from the bootloader.
- **progress_callback** (*TODO, None*) – Optional callback called with the percent of the source_file downloaded. Note, this doesn't include progress of the actual flashing.

Returns Response to a download request, normally nothing.

Return type TODO

Erase(*partition*, *timeout_ms=None*)

Erases the given partition.

Parameters

- **partition** (*TODO*) – Partition to clear.
- **timeout_ms** (*TODO, None*) – TODO

Flash(*partition*, *timeout_ms=0*, *info_cb=<function <lambda>>*)
Flashes the last downloaded file to the given partition.

Parameters

- **partition** (*TODO*) – Partition to overwrite with the new image.
- **timeout_ms** (*int*) – Optional timeout in milliseconds to wait for it to finish.
- **info_cb** (*TODO*) – See `FastbootCommands.Download()`. Usually no messages.

Returns Response to a download request, normally nothing.

Return type TODO

FlashFromFile(*partition*, *source_file*, *source_len=0*, *info_cb=<function <lambda>>*, *progress_callback=None*)
Flashes a partition from the file on disk.

Parameters

- **partition** (*TODO*) – Partition name to flash to.
- **source_file** (*TODO*) – Filename to download to the device.
- **source_len** (*int*) – Optional length of source_file, uses os.stat if not provided.
- **info_cb** (*TODO*) – See Download.
- **progress_callback** (*TODO*) – See Download.

Returns Download and flash responses, normally nothing.

Return type TODO

Getvar(*var*, *info_cb=<function <lambda>>*)
Returns the given variable's definition.

Parameters

- **var** (*TODO*) – A variable the bootloader tracks. Use 'all' to get them all.
- **info_cb** (*TODO*) – See `FastbootCommands.Download()`. Usually no messages.

Returns Value of `var` according to the current bootloader.

Return type TODO

Oem(*command*, *timeout_ms=None*, *info_cb=<function <lambda>>*)
Executes an OEM command on the device.

Parameters

- **command** (*TODO*) – Command to execute, such as 'poweroff' or 'bootconfig read'.
- **timeout_ms** (*TODO, None*) – Optional timeout in milliseconds to wait for a response.
- **info_cb** (*TODO*) – See `FastbootCommands.Download()`. Messages vary based on command.

Returns

Return type The final response from the device.

Reboot(*target_mode=b"*, *timeout_ms=None*)
Reboots the device.

Parameters

- **`target_mode`** (*`bytes`*) – Normal reboot when unspecified. Can specify other target modes such as 'recovery' or 'bootloader'.
- **`timeout_ms`** (*`TODO, None`*) – Optional timeout in milliseconds to wait for a response.

Returns Usually the empty string. Depends on the bootloader and the target_mode.

Return type TODO

`RebootBootloader`(*timeout_ms=None*)

Reboots into the bootloader, usually equiv to Reboot('bootloader').

Parameters **`timeout_ms`** (*`TODO, None`*) – TODO

Returns TODO

Return type TODO

`_FastbootCommands__reset`()

TODO

`_SimpleCommand`(*command*, *arg=None*, ***kwargs*)

TODO

Parameters

- **`command`** (*`TODO`*) – TODO
- **`arg`** (*`TODO, None`*) – TODO
- **`**kwargs`** (*`dict`*) – Keyword arguments
- **`TODO`** – TODO

Returns TODO

Return type TODO

`property usb_handle`

TODO

Returns **self._handle** – TODO

Return type TODO

`exception` `adb.fastboot.`**`FastbootInvalidResponse`**(*message*, **args*)

Bases: *`adb.usb_exceptions.FormatMessageWithArgumentsException`*

Fastboot responded with a header we didn't expect.

`class` `adb.fastboot.`**`FastbootMessage`**(*message*, *header*)

Bases: `tuple`

FastbootMessage

`_fields_defaults = {}`

`class` `adb.fastboot.`**`FastbootProtocol`**(*usb*, *chunk_kb=1024*)

Bases: `object`

Encapsulates the fastboot protocol.

Parameters

- **`usb`** (`adb.common.UsbHandle`) – *`adb.common.UsbHandle`* instance.
- **`chunk_kb`** (*`int`*) – Packet size. For older devices, 4 may be required.

`chunk_kb`

Packet size. For older devices, 4 may be required.

Type int

`usb`

`adb.common.UsbHandle` instance.

Type *adb.common.UsbHandle*

`FINAL_HEADERS = {b'DATA', b'OKAY'}`

`HandleDataSending`(*source_file*, *source_len*, *info_cb=<function <lambda>>*, *progress_callback=None*, *timeout_ms=None*)

Handles the protocol for sending data to the device.

Parameters

- **`source_file`** (*`TODO`*) – File-object to read from for the device.
- **`source_len`** (*`TODO`*) – Amount of data, in bytes, to send to the device.
- **`info_cb`** (*`TODO`*) – Optional callback for text sent from the bootloader.
- **`progress_callback`** (*`TODO, None`*) – Callback that takes the current and the total progress of the current file.
- **`timeout_ms`** (*`TODO, None`*) – Timeout in milliseconds to wait for each response.

Returns OKAY packet's message.

Return type TODO

Raises

- ***`adb.fastboot.FastbootTransferError`*** – When fastboot can't handle this amount of data.
- ***`adb.fastboot.FastbootStateMismatch`*** – Fastboot responded with the wrong packet type.
- ***`adb.fastboot.FastbootRemoteFailure`*** – Fastboot reported failure.
- ***`adb.fastboot.FastbootInvalidResponse`*** – Fastboot responded with an unknown packet type.

`HandleSimpleResponses`(*timeout_ms=None*, *info_cb=<function <lambda>>*)

Accepts normal responses from the device.

Parameters

- **`timeout_ms`** (*`TODO, None`*) – Timeout in milliseconds to wait for each response.
- **`info_cb`** (*`TODO`*) – Optional callback for text sent from the bootloader.

Returns OKAY packet's message.

Return type TODO

SendCommand(*command*, *arg=None*)
Sends a command to the device.

Parameters

- **command** (*str*) – The command to send.
- **arg** (*str*) – Optional argument to the command.

_AcceptResponses(*expected_header*, *info_cb*, *timeout_ms=None*)
Accepts responses until the expected header or a FAIL.

Parameters

- **expected_header** (*TODO*) – `OKAY` or `DATA`
- **info_cb** (*TODO*) – Optional callback for text sent from the bootloader.
- **timeout_ms** (*TODO*) – Timeout in milliseconds to wait for each response.

Returns OKAY packet's message.

Return type TODO

Raises

- ***adb.fastboot.FastbootStateMismatch*** – Fastboot responded with the wrong packet type.
- ***adb.fastboot.FastbootRemoteFailure*** – Fastboot reported failure.
- ***adb.fastboot.FastbootInvalidResponse*** – Fastboot responded with an unknown packet type.

static _HandleProgress(*total*, *progress_callback*)
Calls the callback with the current progress and total.

Parameters

- **total** (*TODO*) – TODO
- **progress_callback** (*TODO*) – TODO

_Write(*data*, *length*, *progress_callback=None*)
Sends the data to the device, tracking progress with the callback.

Parameters

- **data** (*TODO*) – TODO
- **length** (*TODO*) – TODO
- **progress_callback** (*TODO, None*) – TODO

property usb_handle
TODO

Returns **self.usb** – *adb.common.UsbHandle* instance.

Return type *adb.common.UsbHandle*

exception adb.fastboot.**FastbootRemoteFailure**(*message*, **args*)
Bases: *adb.usb_exceptions.FormatMessageWithArgumentsException*

Remote error.

exception `adb.fastboot.`**`FastbootStateMismatch`**(*message*, **args*)

Bases: *`adb.usb_exceptions.FormatMessageWithArgumentsException`*

Fastboot and uboot's state machines are arguing. You Lose.

exception `adb.fastboot.`**`FastbootTransferError`**(*message*, **args*)

Bases: *`adb.usb_exceptions.FormatMessageWithArgumentsException`*

Transfer error.

`adb.fastboot.`**`PROTOCOL = 3`**

From fastboot.c

`adb.fastboot.`**`SUBCLASS = 66`**

From fastboot.c

`adb.fastboot.`**`VENDORS = {1105, 1282, 1478, 2389, 2996, 4046, 6353, 8888, 8980, 16700, 32903`**

From fastboot.c

adb.fastboot_debug module

Fastboot in python.

Call it similar to how you call android's fastboot, but this only accepts usb paths and no serials.

Contents

`adb.fastboot_debug.`**`Devices`**()

Lists the available devices.

```
List of devices attached
015DB7591102001A        device
```

See also:

`adb.fastboot.FastbootCommands.Devices()`

Returns 0

Return type int

`adb.fastboot_debug.`**`_InfoCb`**(*message*)

Print a message to stdout.

Parameters **`message`** (`adb.fastboot.FastbootMessage`) – A *`FastbootMessage`* with `header` and `message` attributes

`adb.fastboot_debug.`**`main`**()

TODO

adb.filesync_protocol module

ADB protocol implementation.

Implements the ADB protocol as seen in android's adb/adbd binaries, but only the host side.

Contents

adb.filesync_protocol.**DEFAULT_PUSH_MODE = 33272**
Default mode for pushed files.

class adb.filesync_protocol.**DeviceFile**(*filename*, *mode*, *size*, *mtime*)
Bases: tuple

_fields_defaults = {}

property filename
Alias for field number 0

property mode
Alias for field number 1

property mtime
Alias for field number 3

property size
Alias for field number 2

class adb.filesync_protocol.**FileSyncConnection**(*adb_connection*, *recv_header_format*)
Bases: object

Encapsulate a FileSync service connection.

Parameters

- **`adb_connection`** (`adb.adb_protocol._AdbConnection`) – ADB connection
- **`recv_header_format`** (*`bytes`*) – TODO

`adb`
ADB connection

Type *adb.adb_protocol._AdbConnection*

`send_buffer`
`bytearray(adb_protocol.MAX_ADB_DATA)` (see *`adb.adb_protocol.MAX_ADB_DATA`*)

Type byte_array

`send_idx`
TODO

Type int

`send_header_len`
`struct.calcsize(b'<2I')`

Type int

`recv_buffer`
TODO

Type bytearray

`recv_header_format`
TODO

Type bytes

`recv_header_len`
`struct.calcsize(recv_header_format)`

Type int

`Read`(*expected_ids*, *read_data=True*)
Read ADB messages and return FileSync packets.

Parameters

- **`expected_ids`** (*`tuple[bytes]`*) – If the received header ID is not in `expected_ids`, an exception will be raised
- **`read_data`** (*`bool`*) – Whether to read the received data

Returns

- **command_id** (*bytes*) – The received header ID
- *tuple* – TODO
- **data** (*bytearray*) – The received data

Raises

- ***`adb.usb_exceptions.AdbCommandFailureException`*** – Command failed
- ***`adb.adb_protocol.InvalidResponseError`*** – Received response was not in `expected_ids`

ReadUntil(*expected_ids*, **finish_ids*)
Useful wrapper around *FileSyncConnection.Read()*.

Parameters

- **expected_ids** (*tuple[bytes]*) – If the received header ID is not in expected_ids, an exception will be raised
- **finish_ids** (*tuple[bytes]*) – We will read until we find a header ID that is in finish_ids

Yields

- **cmd_id** (*bytes*) – The received header ID
- **header** (*tuple*) – TODO
- **data** (*bytearray*) – The received data

Send(*command_id*, *data=b”*, *size=0*)
Send/buffer FileSync packets.

Packets are buffered and only flushed when this connection is read from. All messages have a response from the device, so this will always get flushed.

Parameters

- **command_id** (*bytes*) – Command to send.
- **data** (*str, bytes*) – Optional data to send, must set data or size.
- **size** (*int*) – Optionally override size from len(data).

_CanAddToSendBuffer(*data_len*)
Determine whether data_len bytes of data can be added to the send buffer without exceeding *adb.adb_protocol.MAX_ADB_DATA*.

Parameters data_len (*int*) – The length of the data to be potentially added to the send buffer (not including the length of its header)

Returns Whether data_len bytes of data can be added to the send buffer without exceeding *adb.adb_protocol.MAX_ADB_DATA*

Return type bool

_Flush()
TODO

Raises ***adb.usb_exceptions.WriteFailedError*** – Could not send data

_ReadBuffered(*size*)
Read size bytes of data from self.recv_buffer.

Parameters size (*int*) – The amount of data to read

Returns result – The read data

Return type bytearray

id_to_wire = {b'DATA': 1096040772, b'DENT': 1414415684, b'DONE': 1162760004, b'FAIL':

ids = [b'STAT', b'LIST', b'SEND', b'RECV', b'DENT', b'DONE', b'DATA', b'OKAY', b'FAIL'

wire_to_id = {1096040772: b'DATA', 1145980243: b'SEND', 1162760004: b'DONE', 127986

class `adb.filesync_protocol.`**`FilesyncProtocol`**

Bases: `object`

Implements the FileSync protocol as described in sync.txt.

classmethod **`List`**(*connection*, *path*)

Get a list of the files in `path`.

Parameters

- **`connection`** (`adb.adb_protocol._AdbConnection`) – ADB connection
- **`path`** (*`str, bytes`*) – The path for which we are getting a list of files

Returns **files** – Information about the files in `path`

Return type list[*DeviceFile*]

classmethod **`Pull`**(*connection*, *filename*, *dest_file*, *progress_callback*)

Pull a file from the device into the file-like `dest_file`.

Parameters

- **`connection`** (`adb.adb_protocol._AdbConnection`) – ADB connection
- **`filename`** (*`str`*) – The file to be pulled
- **`dest_file`** (*`_io.BytesIO`*) – File-like object for writing to
- **`progress_callback`** (*`function, None`*) – Callback method that accepts `filename`, `bytes_written`, and `total_bytes`

Raises ***`PullFailedError`*** – Unable to pull file

classmethod **`Push`**(*connection*, *datafile*, *filename*, *st_mode=33272*, *mtime=0*, *progress_callback=None*)

Push a file-like object to the device.

Parameters

- **`connection`** (`adb.adb_protocol._AdbConnection`) – ADB connection
- **`datafile`** (*`_io.BytesIO`*) – File-like object for reading from
- **`filename`** (*`str`*) – Filename to push to
- **`st_mode`** (*`int`*) – Stat mode for filename
- **`mtime`** (*`int`*) – Modification time
- **`progress_callback`** (*`function, None`*) – Callback method that accepts `filename`, `bytes_written`, and `total_bytes`

Raises ***`PushFailedError`*** – Raised on push failure.

static **`Stat`**(*connection*, *filename*)

Get file status (mode, size, and mtime).

Parameters

- **`connection`** (`adb.adb_protocol._AdbConnection`) – ADB connection
- **`filename`** (*`str, bytes`*) – The file for which we are getting info

Returns

- **mode** (*int*) – The mode of the file

- **size** (*int*) – The size of the file
- **mtime** (*int*) – The time of last modification for the file

Raises ***adb.adb_protocol.InvalidResponseError*** – Expected STAT response to STAT, got something else

classmethod _HandleProgress(*progress_callback*)
Calls the callback with the current progress and total bytes written/received.

Parameters **progress_callback** (*function*) – Callback method that accepts `filename`, `bytes_written`, and `total_bytes`; total_bytes will be -1 for file-like objects.

exception adb.filesync_protocol.**InterleavedDataError**
Bases: `Exception`

We only support command sent serially.

exception adb.filesync_protocol.**InvalidChecksumError**
Bases: `Exception`

Checksum of data didn't match expected checksum.

adb.filesync_protocol.**MAX_PUSH_DATA = 2048**
Maximum size of a filesync DATA packet.

exception adb.filesync_protocol.**PullFailedError**
Bases: `Exception`

Pulling a file failed for some reason.

exception adb.filesync_protocol.**PushFailedError**
Bases: `Exception`

Pushing a file failed for some reason.

adb.sign_cryptography module

ADB authentication using the `cryptography` package.

Contents

class adb.sign_cryptography.**CryptographySigner**(*rsa_key_path*)
Bases: *adb.adb_protocol.AuthSigner*

AuthSigner using cryptography.io.

> **Warning:** This is currently broken!

Parameters **`rsa_key_path`** (*`str`*) – The path to the private key.

`public_key`
The contents of the public key file

Type str

`rsa_key`
The loaded private key

Type cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey

`GetPublicKey`()
Returns the public key in PEM format without headers or newlines.

Returns **self.public_key** – The contents of the public key file

Return type str

`Sign`(*data*)
Signs given data using a private key.

Parameters **`data`** (*`TODO`*) – TODO

Returns The signed `data`

Return type TODO

adb.sign_pycryptodome module

ADB authentication using `pycryptodome`.

Contents

`class` `adb.sign_pycryptodome.`**`PycryptodomeAuthSigner`**(*rsa_key_path=None*)
Bases: *`adb.adb_protocol.AuthSigner`*

TODO

Parameters **`rsa_key_path`** (*`str, None`*) – The path to the private key

`public_key`
The contents of the public key file

Type str

`rsa_key`
The contents of theprivate key

Type Crypto.PublicKey.RSA.RsaKey

`GetPublicKey`()
Returns the public key in PEM format without headers or newlines.

Returns **self.public_key** – The contents of the public key file

Return type str

Sign (*data*)
Signs given data using a private key.

Parameters **data** (*bytes, bytearray*) – The data to be signed

Returns The signed `data`

Return type bytes

adb.sign_pythonrsa module

adb.usb_exceptions module

Common exceptions for ADB and Fastboot.

Contents

exception adb.usb_exceptions.**AdbCommandFailureException**
Bases: `Exception`

ADB Command returned a FAIL.

exception adb.usb_exceptions.**AdbOperationException**
Bases: `Exception`

Failed to communicate over adb with device after multiple retries.

exception adb.usb_exceptions.**CommonUsbError**
Bases: `Exception`

Base class for usb communication errors.

exception adb.usb_exceptions.**DeviceAuthError**(*message*, **args*)

Bases: *adb.usb_exceptions.FormatMessageWithArgumentsException*

Device authentication failed.

exception adb.usb_exceptions.**DeviceNotFoundError**(*message*, **args*)

Bases: *adb.usb_exceptions.FormatMessageWithArgumentsException*

Device isn't on USB.

exception adb.usb_exceptions.**FormatMessageWithArgumentsException**(*message*, **args*)

Bases: *adb.usb_exceptions.CommonUsbError*

Exception that both looks good and is functional.

Okay, not that kind of functional, it's still a class.

This interpolates the message with the given arguments to make it human-readable, but keeps the arguments in case other code try-excepts it.

Parameters

- **message** (*str*) – The error message
- **args** (*str*) – Positional arguments for formatting `message`

exception adb.usb_exceptions.**LibusbWrappingError**(*msg*, *usb_error*)

Bases: *adb.usb_exceptions.CommonUsbError*

Wraps `libusb1` errors while keeping their original usefulness.

Parameters

- **msg** (*str*) – The error message
- **usb_error** (*libusb1.USBError*) – An exception from `libusb1`

usb_error

An exception from `libusb1`

Type libusb1.USBError

exception adb.usb_exceptions.**ReadFailedError**(*msg*, *usb_error*)

Bases: *adb.usb_exceptions.LibusbWrappingError*

Raised when the device doesn't respond to our commands.

exception adb.usb_exceptions.**TcpTimeoutException**(*message*, **args*)

Bases: *adb.usb_exceptions.FormatMessageWithArgumentsException*

TCP connection timed out in the time out given.

exception adb.usb_exceptions.**WriteFailedError**(*msg*, *usb_error*)

Bases: *adb.usb_exceptions.LibusbWrappingError*

Raised when the device doesn't accept our command.

1.1.2 Module contents

Note: This is not an official Google project. It is maintained by ex-Google engineers. For a better maintained option, look at adb_shell.

This repository contains a pure-python implementation of the ADB and Fastboot protocols, using libusb1 for USB communications.

This is a complete replacement and rearchitecture of the Android project's ADB and fastboot code.

This code is mainly targeted to users that need to communicate with Android devices in an automated fashion, such as in automated testing. It does not have a daemon between the client and the device, and therefore does not support multiple simultaneous commands to the same device. It does support any number of devices and *never* communicates with a device that it wasn't intended to, unlike the Android project's ADB.

CHAPTER TWO

USING AS STANDALONE TOOL

Install using pip:

```
pip install adb
```

Once installed, two new binaries should be available: `pyadb` and `pyfastboot`.

```
pyadb devices
pyadb shell ls /sdcard
```

Running `./make_tools.py` creates two files: `adb.zip` and `fastboot.zip`. They can be run similar to native `adb` and `fastboot` via the python interpreter:

```
python adb.zip devices
python adb.zip shell ls /sdcard
```

CHAPTER

THREE

USING AS A PYTHON LIBRARY

A presentation was made at PyCon 2016, and here's some demo code:

```
import os.path as op

from adb import adb_commands
from adb import sign_cryptography


# KitKat+ devices require authentication
signer = sign_cryptography.CryptographySigner(
    op.expanduser('~/.android/adbkey'))
# Connect to the device
device = adb_commands.AdbCommands()
device.ConnectDevice(
    rsa_keys=[signer])
# Now we can use Shell, Pull, Push, etc!
for i in xrange(10):
    print device.Shell('echo %d' % i)
```

CHAPTER

FOUR

PROS

- Simpler code due to use of libusb1 and Python.
- API can be used by other Python code easily.
- Errors are propagated with tracebacks, helping debug connectivity issues.
- No daemon outliving the command.
- Can be packaged as standalone zips that can be run independent of the CPU architecture (e.g. x86 vs ARM).

CHAPTER FIVE

CONS

- Technically slower due to Python, mitigated by no daemon.
- Only one command per device at a time.
- More dependencies than Android's ADB.

CHAPTER

SIX

DEPENDENCIES

- libusb1 (1.0.16+)
- python-libusb1 (1.2.0+)
- **adb.zip**: one of
  - py-cryptography
  - python-rsa (3.2+)
- **fastboot.zip** (optional)
  - python-progressbar (2.3+)

CHAPTER SEVEN

HISTORY

7.1 1.0.0

- Initial version

7.2 1.1.0

- Added TcpHandle (jameyhicks)
- Various timing and other changes (alusco)

7.3 1.2.0

- Update to libusb1 1.6+ (bytearray output)
- Add support for Python 3.6
- Create adb.zip and fastboot.zip as executable tools.
- Add Travis CI integration
- Support multiple crypto libraries (M2Crypto + python-rsa)
- Push directories

7.4 1.3.0

7.4.1 Backwards Incompatible changes

`adb_commands.AdbCommands` is now a normal class rather than a collection of staticmethods. Using the following example code to get started:

```
device = adb_commands.AdbCommands()
device.ConnectDevice(rsa_keys=[signer])
```

7.4.2 Other changes/fixes

Many changes since 1.2.0!

- New entrypoints exposed by pip: pyadb and pyfastboot
- Lots of Python 2/3 compatibility fixes
- Windows compatibility fixes
- Transfer progress available (`Push`, `Pull`, `Install`)
- Handle some misbehaving devices (double CLSE bug)
- New options for `Push` and `Install` (`st_mode` and `grant_permissions`)

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX