# **ADB Shell Documentation**

Release 0.4.0

**Jeff Irion** 

## **CONTENTS**

l	adb_shell	1	
	1.1 adb_shell package	1	
	Installation	55	
	2.1 Async	55	
	2.2 USB Support (Experimental)	55	
3	Example Usage	57	
	3.1 Generate ADB Key Files	57	
4 Indices and tables			
Рy	ython Module Index	61	
In	dex	63	

**CHAPTER** 

ONE

## ADB\_SHELL

## 1.1 adb shell package

## 1.1.1 Subpackages

adb\_shell.auth package

**Submodules** 

## adb\_shell.auth.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
    uint32_t exponent;
} RSAPublicKey;
```

## **Contents**

- \_to\_bytes()
- decode\_pubkey()
- decode\_pubkey\_file()

```
• encode pubkev()
   • get_user_info()
   • keygen()
   • write_public_keyfile()
adb shell.auth.keygen.ANDROID PUBKEY MODULUS SIZE = 256
    Size of an RSA modulus such as an encrypted block or a signature.
adb_shell.auth.keygen.ANDROID_PUBKEY_MODULUS_SIZE_WORDS = 64
    Size of the RSA modulus in words.
adb_shell.auth.keygen.ANDROID_RSAPUBLICKEY_STRUCT = '<LL256s256sL'
    Python representation of "struct RSAPublicKey"
adb_shell.auth.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_shell.auth.keygen.decode_pubkey(public_key)
    Decode a public RSA key stored in Android's custom binary format.
         Parameters public_key (TODO) - TODO
adb_shell.auth.keygen.decode_pubkey_file(public_key_path)
    TODO
         Parameters public_key_path(str)-TODO
adb_shell.auth.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_shell.auth.keygen.get_user_info()
    TODO
         Returns ' <username>@<hostname>
         Return type str
adb_shell.auth.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\_shell.auth.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 shell.auth.sign cryptography module

ADB authentication using the cryptography package.

## **Contents**

- CryptographySigner
  - CryptographySigner.GetPublicKey()
  - CryptographySigner.Sign()

```
class adb_shell.auth.sign_cryptography.CryptographySigner(rsa_key_path)
```

Bases: object

AuthSigner using cryptography.io.

**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\_shell.auth.sign\_pycryptodome module

ADB authentication using pycryptodome.

## **Contents**

```
• PycryptodomeAuthSigner
       - PycryptodomeAuthSigner.GetPublicKey()
       - PycryptodomeAuthSigner.Sign()
class adb_shell.auth.sign_pycryptodome.PycryptodomeAuthSigner(rsa_key_path=None)
     Bases: object
     AuthSigner using the pycryptodome package.
          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\_shell.auth.sign\_pythonrsa module

ADB authentication using the rsa package.

## **Contents**

```
_Accum
_Accum.digest()
_Accum.update()
_load_rsa_private_key()
PythonRSASigner
_PythonRSASigner.FromRSAKeyPath()
_PythonRSASigner.GetPublicKey()
_PythonRSASigner.Sign()
```

```
class adb_shell.auth.sign_pythonrsa.PythonRSASigner(pub=None, priv=None)
    Bases: object
```

Implements adb\_protocol.AuthSigner using http://stuvel.eu/rsa.

#### **Parameters**

- pub (str, None) The contents of the public key file
- priv (str, None) The path to the private key

## priv\_key

The loaded private key

**Type** rsa.key.PrivateKey

#### pub\_key

The contents of the public key file

Type str, None

## classmethod FromRSAKeyPath (rsa\_key\_path)

Create a PythonRSASigner instance using the provided private key.

**Parameters** rsa\_key\_path (str) - The path to the private key; the public key must be rsa\_key\_path + '.pub'.

**Returns** A *PythonRSASigner* with private key rsa\_key\_path and public key rsa\_key\_path + '.pub'

Return type PythonRSASigner

## GetPublicKey()

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

**Returns** self.pub\_key – The contents of the public key file, or None if a public key was not provided.

Return type str, None

#### Sign (data)

Signs given data using a private key.

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

Returns The signed data

Return type bytes

```
\textbf{class} \ \texttt{adb\_shell.auth.sign\_pythonrsa.\_Accum}
```

Bases: object

A fake hashing algorithm.

The Python rsa lib hashes all messages it signs. ADB does it already, we just need to slap a signature on top of already hashed message. Introduce a "fake" hashing algo for this.

#### buf

A buffer for storing data before it is signed

Type bytes

## digest()

Return the digest value as a string of binary data.

Returns self. buf - self. buf

#### Return type bytes

```
update (msg)
```

Update this hash object's state with the provided msg.

**Parameters** msg (bytes) - The message to be appended to self.\_buf

```
adb_shell.auth.sign_pythonrsa._load_rsa_private_key(pem)
```

PEM encoded PKCS#8 private key -> rsa.PrivateKey.

ADB uses private RSA keys in pkcs#8 format. The rsa library doesn't support them natively. Do some ASN unwrapping to extract naked RSA key (in der-encoded form).

#### See:

- https://www.ietf.org/rfc/rfc2313.txt
- http://superuser.com/a/606266

**Parameters** pem(str) – The private key to be loaded

**Returns** The loaded private key

Return type rsa.key.PrivateKey

#### **Module contents**

## adb shell.transport package

#### **Submodules**

## adb shell.transport.base transport module

A base class for transports used to communicate with a device.

- BaseTransport
  - BaseTransport.bulk\_read()
  - BaseTransport.bulk\_write()
  - BaseTransport.close()
  - BaseTransport.connect()

class adb\_shell.transport.base\_transport.BaseTransport

Bases: abc.ABC

A base transport class.

```
_abc_impl = <_abc_data object>
```

abstract bulk\_read(numbytes, transport\_timeout\_s)

Read data from the device.

## **Parameters**

- numbytes (int) The maximum amount of data to be received
- transport\_timeout\_s (float, None) A timeout for the read operation

Returns The received data

Return type bytes

```
abstract bulk_write(data, transport_timeout_s)
```

Send data to the device.

#### **Parameters**

- data (bytes) The data to be sent
- transport\_timeout\_s (float, None) A timeout for the write operation

**Returns** The number of bytes sent

Return type int

## abstract close()

Close the connection.

## abstract connect(transport\_timeout\_s)

Create a connection to the device.

Parameters transport\_timeout\_s (float, None) - A connection timeout

#### adb shell.transport.base transport async module

A base class for transports used to communicate with a device.

- BaseTransportAsync
  - BaseTransportAsync.bulk\_read()
  - BaseTransportAsync.bulk write()
  - BaseTransportAsync.close()
  - BaseTransportAsync.connect()

 ${\tt class} \verb| adb\_shell.transport.base\_transport\_async. \verb| BaseTransportAsync| \\$ 

Bases: abc.ABC

A base transport class.

```
_abc_impl = <_abc_data object>
```

abstract async bulk\_read(numbytes, transport\_timeout\_s)

Read data from the device.

#### **Parameters**

- numbytes (int) The maximum amount of data to be received
- transport timeout s(float, None) A timeout for the read operation

Returns The received data

Return type bytes

abstract async bulk\_write(data, transport\_timeout\_s)

Send data to the device.

#### **Parameters**

- data (bytes) The data to be sent
- transport\_timeout\_s (float, None) A timeout for the write operation

**Returns** The number of bytes sent

Return type int

```
abstract async close()
          Close the connection.
     abstract async connect(transport_timeout_s)
          Create a connection to the device.
              Parameters transport_timeout_s (float, None) - A connection timeout
adb_shell.transport.tcp_transport module
A class for creating a socket connection with the device and sending and receiving data.
   • TcpTransport
       - TcpTransport.bulk_read()
       - TcpTransport.bulk_write()
       - TcpTransport.close()
       - TcpTransport.connect()
class adb_shell.transport.tcp_transport.TcpTransport (host, port=5555)
     Bases: adb_shell.transport.base_transport.BaseTransport
     TCP connection object.
          Parameters
                • host (str) – The address of the device; may be an IP address or a host name
                • port (int) – The device port to which we are connecting (default is 5555)
     _connection
          A socket connection to the device
              Type socket.socket, None
     _host
          The address of the device; may be an IP address or a host name
              Type str
     port
          The device port to which we are connecting (default is 5555)
              Type int
     abc impl = < abc data object>
     bulk_read (numbytes, transport_timeout_s)
          Receive data from the socket.
              Parameters
                  • numbytes (int) - The maximum amount of data to be received
                  • transport_timeout_s (float, None) - When the timeout argument is omitted,
                    select.select blocks until at least one file descriptor is ready. A time-out value of
                   zero specifies a poll and never blocks.
              Returns The received data
              Return type bytes
              Raises TcpTimeoutException – Reading timed out.
```

```
bulk_write (data, transport_timeout_s)
```

Send data to the socket.

#### **Parameters**

- data (bytes) The data to be sent
- transport\_timeout\_s (float, None) When the timeout argument is omitted, select.select blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

Returns The number of bytes sent

Return type int

Raises TcpTimeoutException - Sending data timed out. No data was sent.

close()

Close the socket connection.

connect (transport\_timeout\_s)

Create a socket connection to the device.

Parameters transport\_timeout\_s (float, None) - Set the timeout on the socket instance

## adb shell.transport.tcp transport async module

A class for creating a socket connection with the device and sending and receiving data.

- TcpTransportAsync
  - TcpTransportAsync.bulk\_read()
  - TcpTransportAsync.bulk\_write()
  - TcpTransportAsync.close()
  - TcpTransportAsync.connect()

```
class adb_shell.transport.tcp_transport_async.TcpTransportAsync(host,
```

port=5555)

Bases: adb\_shell.transport.base\_transport\_async.BaseTransportAsync

TCP connection object.

#### **Parameters**

- host (str) The address of the device; may be an IP address or a host name
- port (int) The device port to which we are connecting (default is 5555)

\_host

The address of the device; may be an IP address or a host name

Type str

\_port

The device port to which we are connecting (default is 5555)

Type int

\_reader

Object for reading data from the socket

Type StreamReader, None

#### writer

Object for writing data to the socket

Type StreamWriter, None

```
_abc_impl = <_abc_data object>
```

async bulk\_read(numbytes, transport\_timeout\_s)

Receive data from the socket.

#### **Parameters**

- numbytes (int) The maximum amount of data to be received
- transport\_timeout\_s (float, None) Timeout for reading data from the socket; if it is None, then it will block until the read operation completes

**Returns** The received data

Return type bytes

Raises TcpTimeoutException - Reading timed out.

async bulk\_write(data, transport\_timeout\_s)

Send data to the socket.

#### **Parameters**

- data (bytes) The data to be sent
- transport\_timeout\_s (float, None) Timeout for writing data to the socket; if it is None, then it will block until the write operation completes

Returns The number of bytes sent

Return type int

Raises TcpTimeoutException - Sending data timed out. No data was sent.

async close()

Close the socket connection.

async connect (transport\_timeout\_s)

Create a socket connection to the device.

**Parameters transport\_timeout\_s** (*float*, *None*) – Timeout for connecting to the socket; if it is None, then it will block until the operation completes

## adb\_shell.transport.usb\_transport module

A class for creating a USB connection with the device and sending and receiving data.

**Warning:** USB support is an experimental feature.

- get interface()
- interface\_matcher()
- UsbTransport
  - UsbTransport.\_find()
  - UsbTransport.\_find\_and\_open()

```
- UsbTransport._find_devices()
       - UsbTransport._find_first()
       - UsbTransport._flush_buffers()
       - UsbTransport._open()
       - UsbTransport._port_path_matcher()
       - UsbTransport. serial matcher()
       - UsbTransport._timeout()
       - UsbTransport.bulk_read()
       - UsbTransport.bulk_write()
       - UsbTransport.close()
       - UsbTransport.connect()
       - UsbTransport.port_path
       - UsbTransport.serial number
       - UsbTransport.usb info
adb_shell.transport.usb_transport.DEFAULT_TIMEOUT_S = 10
    Default timeout
class adb_shell.transport.usb_transport.UsbTransport(device,
                                                                                 setting,
                                                                                    de-
                                                              fault_transport_timeout_s=None)
    Bases: adb_shell.transport.base_transport.BaseTransport
    USB communication object. Not thread-safe.
    Handles reading and writing over USB with the proper endpoints, exceptions, and interface claiming.
         Parameters
              • device (usb1.USBDevice) – libush device to connect to.
              • setting (usb1.USBInterfaceSetting) - libusb setting with the correct endpoints
                to communicate with.
              • usb info (TODO, None) - String describing the usb path/serial/device, for debugging.
              • default transport timeout s(TODO, None) - Timeout in seconds for all I/O.
    _default_transport_timeout_s
         Timeout in seconds for all I/O.
            Type TODO, None
     device
         libusb_device to connect to.
            Type TODO
     transport
         TODO
            Type TODO
    _interface_number
         TODO
```

12

```
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
_usb_info
    String describing the usb path/serial/device, for debugging.
        Type TODO
write endpoint
    TODO
        Type TODO, None
_HANDLE_CACHE = <WeakValueDictionary>
_HANDLE_CACHE_LOCK = <unlocked _thread.lock object>
_abc_impl = <_abc_data object>
classmethod _find(setting_matcher,
                                         port_path=None,
                                                              serial=None,
                                                                                de-
                     fault_transport_timeout_s=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
           • default_transport_timeout_s (TODO, None) - TODO
        Returns TODO
        Return type TODO
classmethod _find_and_open (setting_matcher,
                                                port_path=None,
                                                                  serial=None.
                                                                                de-
                               fault_transport_timeout_s=None)
    TODO
        Parameters
           • setting_matcher (TODO) - TODO
           • port_path (TODO, None) - TODO
           • serial (TODO, None) - TODO
           • default_transport_timeout_s (TODO, None) - TODO
        Returns dev - TODO
        Return type TODO
```

#### **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 UsbTransport is valid. None to match any device.
- **usb\_info** (*str*) Info string describing device(s).
- **default\_transport\_timeout\_s** (*TODO*, *None*) Default timeout of commands in seconds.

**Yields** *TODO* – UsbTransport instances

#### **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*) Function that returns True if the given UsbTransport is valid. None to match any device.
- **usb\_info** (*str*) Info string describing device(s).
- **default\_transport\_timeout\_s** (*TODO*, *None*) **Default** timeout of commands in seconds.

**Returns** An instance of *UsbTransport* 

Return type TODO

**Raises** adb\_shell.exceptions.DeviceNotFoundError - Raised if the device is not available.

```
_flush_buffers()
TODO
```

Raises adb\_shell.exceptions.UsbReadFailedError-TODO

\_open()

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

## classmethod \_port\_path\_matcher(port\_path)

Returns a device matcher for the given port path.

Parameters port path (TODO) - TODO

Returns TODO

Return type function

## classmethod \_serial\_matcher(serial)

Returns a device matcher for the given serial.

Parameters serial (TODO) - TODO

Returns TODO

## Return type function

```
_timeout_ms (transport_timeout_s)
TODO
```

Returns TODO

Return type TODO

bulk\_read (numbytes, transport\_timeout\_s=None)

Receive data from the USB device.

#### **Parameters**

- numbytes (int) The maximum amount of data to be received
- transport\_timeout\_s (float, None) When the timeout argument is omitted, select.select blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

Returns The received data

Return type bytes

Raises adb\_shell.exceptions.UsbReadFailedError - Could not receive data

bulk\_write (data, transport\_timeout\_s=None)

Send data to the USB device.

#### **Parameters**

- data (bytes) The data to be sent
- transport\_timeout\_s (float, None) When the timeout argument is omitted, select.select blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

Returns The number of bytes sent

Return type int

#### Raises

- adb\_shell.exceptions.UsbWriteFailedError This transport has been closed, probably due to another being opened
- adb\_shell.exceptions.UsbWriteFailedError Could not send data

close()

Close the USB connection.

connect (transport timeout s=None)

Create a USB connection to the device.

Parameters transport\_timeout\_s (float, None) - Set the timeout on the USB instance

classmethod find\_adb (serial=None, port\_path=None, default\_transport\_timeout\_s=None)
TODO

#### **Parameters**

- serial (TODO) TODO
- port\_path (TODO) TODO

• **default\_transport\_timeout\_s** (*TODO*, *None*) – Default timeout of commands in seconds.

Returns TODO

Return type UsbTransport

classmethod find\_all\_adb\_devices (default\_transport\_timeout\_s=None)

Find all ADB devices attached via USB.

**Parameters default\_transport\_timeout\_s** (*TODO*, *None*) – Default timeout of commands in seconds.

**Returns** A generator which yields each ADB device attached via USB.

Return type generator

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\_shell.transport.usb\_transport.get\_interface(setting)

Get the class, subclass, and protocol for the given USB setting.

Parameters setting (TODO) - TODO

## Returns

- TODO TODO
- TODO TODO
- TODO TODO

adb\_shell.transport.usb\_transport.interface\_matcher(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

## **Module contents**

## 1.1.2 Submodules

## adb\_shell.adb\_device module

Implement the AdbDevice class, which can connect to a device and run ADB shell commands.

#### **Contents**

```
• _AdbIOManager
   - _AdbIOManager._read_bytes_from_device()
   - _AdbIOManager._read_expected_packet_from_device()
   - _AdbIOManager._read_packet_from_device()
   - AdbIOManager. send()
   - _AdbIOManager.close()
   - _AdbIOManager.connect()
   - _AdbIOManager.read()
   - _AdbIOManager.send()
• AdbDevice
   - AdbDevice._clse()
   - AdbDevice._filesync_flush()
   - AdbDevice._filesync_read()
   - AdbDevice. filesync read buffered()
   - AdbDevice._filesync_read_until()
   - AdbDevice._filesync_send()
   - AdbDevice._okay()
   - AdbDevice._open()
   - AdbDevice._pull()
   - AdbDevice._push()
   - AdbDevice._read_until()
   - AdbDevice._read_until_close()
   - AdbDevice._service()
   - AdbDevice._streaming_command()
   - AdbDevice. streaming service()
   - AdbDevice.available
   - AdbDevice.close()
```

- AdbDevice.connect()

- AdbDevice.list()
- AdbDevice.max chunk size
- AdbDevice.pull()
- AdbDevice.push()
- AdbDevice.root()
- AdbDevice.shell()
- AdbDevice.stat()
- AdbDevice.streaming\_shell()
- AdbDeviceTcp
- AdbDeviceUsb

class adb\_shell.adb\_device.AdbDevice(transport, default\_transport\_timeout\_s=None, banner=None)

Bases: object

A class with methods for connecting to a device and executing ADB commands.

#### **Parameters**

- **transport** (BaseTransport) A user-provided transport for communicating with the device; must be an instance of a subclass of BaseTransport
- default\_transport\_timeout\_s (float, None) Default timeout in seconds for transport packets, or None
- banner (str, bytes, None) The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via socket. gethostname()

Raises adb\_shell.exceptions.InvalidTransportError - The passed transport is not an instance of a subclass of BaseTransport

#### available

Whether an ADB connection to the device has been established

Type bool

## \_banner

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

## \_default\_transport\_timeout\_s

Default timeout in seconds for transport packets, or None

Type float, None

## \_io\_manager

Used for handling all ADB I/O

**Type** \_*AdbIOManager* 

## \_local\_id

The local ID that is used for ADB transactions; the value is incremented each time and is always in the range  $[1, 2^32)$ 

Type int

#### local id lock

A lock for protecting \_local\_id; this is never held for long

Type Lock

#### maxdata

Maximum amount of data in an ADB packet

**Type** int

## \_clse(adb\_info)

Send a b'CLSE' message and then read a b'CLSE' message.

**Warning:** This is not to be confused with the AdbDevice.close() method!

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

#### \_filesync\_flush(adb\_info, filesync\_info)

Write the data in the buffer up to filesync\_info.send\_idx, then set filesync\_info.send idx to 0.

#### **Parameters**

- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

## \_filesync\_read(expected\_ids, adb\_info, filesync\_info)

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
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

#### Returns

- **command\_id** (*bytes*) The received header ID
- tuple The contents of the header
- data (bytearray, None) The received data, or None if the command ID is adb\_shell. constants.STAT

#### Raises

- adb\_shell.exceptions.AdbCommandFailureException Command failed
- adb\_shell.exceptions.InvalidResponseError Received response was not in expected\_ids

## \_filesync\_read\_buffered(size, adb\_info, filesync\_info)

Read size bytes of data from self.recv\_buffer.

#### **Parameters**

- **size** (*int*) The amount of data to read
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

Returns result – The read data

**Return type** bytearray

\_filesync\_read\_until (expected\_ids, finish\_ids, adb\_info, filesync\_info)
Useful wrapper around AdbDevice.\_filesync\_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
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

### **Yields**

- cmd\_id (bytes) The received header ID
- header (tuple) TODO
- data (bytearray) The received data

\_filesync\_send (command\_id, adb\_info, filesync\_info, data=b", size=None) 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.
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- $\bullet$  filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction
- data (str, bytes) Optional data to send, must set data or size.
- **size** (*int*, *None*) Optionally override size from len(data).

## \_get\_transport\_timeout\_s (transport\_timeout\_s)

Use the provided transport\_timeout\_s if it is not None; otherwise, use self.  $\_default\_transport\_timeout\_s$ 

 $\textbf{Parameters transport\_timeout\_s} \ (\textit{float, None}) - The \ potential \ transport \ timeout$ 

Returns transport\_timeout\_s if it is not None; otherwise, self. \_default\_transport\_timeout\_s

## Return type float

```
okay (adb info)
```

Send an b'OKAY' mesage.

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

**\_open** (destination, transport\_timeout\_s, read\_timeout\_s, timeout\_s)

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

- 1. send() an b'OPEN' command to the device that specifies the local id
- 2. read() the response from the device and fill in the adb\_info.remote\_id attribute

## **Parameters**

- destination (bytes) b'SERVICE: COMMAND'
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish

Returns adb\_info - Info and settings for this ADB transaction

**Return type** \_*AdbTransactionInfo* 

\_pull (device\_path, stream, progress\_callback, adb\_info, filesync\_info)
Pull a file from the device into the file-like local\_path.

#### **Parameters**

- **device\_path** (str) The file on the device that will be pulled
- **stream** (\_io.BytesIO) File-like object for writing to
- progress\_callback (function, None) Callback method that accepts
  device\_path, bytes\_written, and total\_bytes
- $adb\_info$  (\_AdbTransactionInfo) Info and settings for this ADB transaction
- **filesync\_info** (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

\_push (stream, device\_path, st\_mode, mtime, progress\_callback, adb\_info, filesync\_info)

Push a file-like object to the device.

#### **Parameters**

- **stream** (\_io.BytesIO) File-like object for reading from
- **device\_path** (str) Destination on the device to write to
- **st\_mode** (*int*) Stat mode for the file
- mtime (int) Modification time
- progress\_callback (function, None) Callback method that accepts device\_path, bytes\_written, and total\_bytes
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

Raises PushFailedError - Raised on push failure.

#### \_read\_until (expected\_cmds, adb\_info)

Read a packet, acknowledging any write packets.

- 1. Read data via \_AdbIOManager.read()
- 2. If a b'WRTE' packet is received, send an b'OKAY' packet via AdbDevice.\_okay()
- 3. Return the cmd and data that were read by \_AdbIOManager.read()

#### **Parameters**

- expected\_cmds (list[bytes]) \_AdbIOManager.read() will look for a packet whose command is in expected\_cmds
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### Returns

- **cmd** (*bytes*) The command that was received by \_AdbIOManager.read(), which is in adb\_shell.constants.WIRE\_TO\_ID and must be in expected\_cmds
- data (bytes) The data that was received by \_AdbIOManager.read()

#### \_read\_until\_close(adb\_info)

Yield packets until a b 'CLSE' packet is received.

- 1. Read the cmd and data fields from a b'CLSE' or b'WRTE' packet via AdbDevice. \_read\_until()
- 2. If cmd is b'CLSE', then send a b'CLSE' message and stop
- 3. Yield data and repeat

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

Yields data (bytes) - The data that was read by AdbDevice.\_read\_until()

\_service (service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True)
Send an ADB command to the device.

#### **Parameters**

- **service** (bytes) The ADB service to talk to (e.g., b'shell')
- command (bytes) The command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **decode** (bool) Whether to decode the output to utf8 before returning

**Returns** The output of the ADB command as a string if decode is True, otherwise as bytes.

Return type bytes, str

- \_streaming\_command(service, command, transport\_timeout\_s, read\_timeout\_s, timeout\_s)
  One complete set of packets for a single command.
  - \_open() a new connection to the device, where the destination parameter is service:command
  - 2. Read the response data via AdbDevice.\_read\_until\_close()

**Note:** All the data is held in memory, and thus large responses will be slow and can fill up memory.

#### **Parameters**

- **service** (bytes) The ADB service (e.g., b'shell', as used by AdbDevice. shell())
- command (bytes) The service command
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish

**Yields** bytes – The responses from the service.

\_streaming\_service (service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, decode=True)

Send an ADB command to the device, yielding each line of output.

#### **Parameters**

- **service** (bytes) The ADB service to talk to (e.g., b'shell')
- command (bytes) The command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- **decode** (bool) Whether to decode the output to utf8 before returning

**Yields** bytes, str – The line-by-line output of the ADB command as a string if decode is True, otherwise as bytes.

## property available

Whether or not an ADB connection to the device has been established.

 $\textbf{Returns} \text{ self.} \underline{\hspace{0.1cm}} \text{available}$ 

Return type bool

## close()

Close the connection via the provided transport's close () method.

Establish an ADB connection to the device.

See\_AdbIOManager.connect().

#### **Parameters**

- rsa\_keys (list, None) A list of signers of type CryptographySigner, PycryptodomeAuthSigner, or PythonRSASigner
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- auth\_timeout\_s (float, None) The time in seconds to wait for a b'CNXN' authentication response
- read\_timeout\_s (float) The total time in seconds to wait for expected commands in \_AdbIOManager.\_read\_expected\_packet\_from\_device()
- auth\_callback (function, None) Function callback invoked when the connection needs to be accepted on the device

**Returns** Whether the connection was established (AdbDevice.available)

#### Return type bool

exec\_out (command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True)

Send an ADB exec-out command to the device.

https://www.linux-magazine.com/Issues/2017/195/Ask-Klaus

## **Parameters**

- command(str) The exec-out command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **decode** (bool) Whether to decode the output to utf8 before returning

**Returns** The output of the ADB exec-out command as a string if decode is True, otherwise as bytes.

## Return type bytes, str

**list** (*device\_path*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*) Return a directory listing of the given path.

#### **Parameters**

- **device\_path** (str) Directory to list.
- transport\_timeout\_s (float, None) Expected timeout for any part of the pull.

• read\_timeout\_s (float) - The total time in seconds to wait for a b'CLSE' or b'OKAY' command in AdbIOManager.read()

**Returns** files – Filename, mode, size, and mtime info for the files in the directory

**Return type** list[*DeviceFile*]

#### property max\_chunk\_size

Maximum chunk size for filesync operations

**Returns** Minimum value based on adb\_shell.constants.MAX\_CHUNK\_SIZE and \_max\_data / 2, fallback to legacy adb\_shell.constants.MAX\_PUSH\_DATA

## Return type int

pull (device\_path, local\_path, progress\_callback=None, transport\_timeout\_s=None,
 read\_timeout\_s=10.0)
Pull a file from the device.

#### **Parameters**

- **device\_path** (str) The file on the device that will be pulled
- local\_path (str) The path to where the file will be downloaded
- progress\_callback (function, None) Callback method that accepts device path, bytes written, and total bytes
- transport\_timeout\_s (float, None) Expected timeout for any part of the pull.
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in AdbIOManager.read()

push (local\_path, device\_path, st\_mode=33272, mtime=0, progress\_callback=None, transport\_timeout\_s=None, read\_timeout\_s=10.0) Push a file or directory to the device.

#### **Parameters**

- **local\_path** (*str*) Either a filename or a directory to push to the device.
- **device** path (str) Destination on the device to write to.
- **st\_mode** (int) Stat mode for local\_path
- **mtime** (*int*) Modification time to set on the file.
- progress\_callback (function, None) Callback method that accepts
  device\_path, bytes\_written, and total\_bytes
- transport\_timeout\_s (float, None) Expected timeout for any part of the push.
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()

root (transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None)
Gain root access.

The device must be rooted in order for this to work.

## **Parameters**

• transport\_timeout\_s (float, None) - Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()

- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in AdbIOManager.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **shell** (*command*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*, *timeout\_s=None*, *decode=True*) Send an ADB shell command to the device.

#### **Parameters**

- **command** (str) The shell command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **decode** (bool) Whether to decode the output to utf8 before returning

**Returns** The output of the ADB shell command as a string if decode is True, otherwise as bytes.

## Return type bytes, str

**stat** (device\_path, transport\_timeout\_s=None, read\_timeout\_s=10.0)
Get a file's stat () information.

## **Parameters**

- **device\_path** (str) The file on the device for which we will get information.
- transport\_timeout\_s (float, None) Expected timeout for any part of the pull.
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()

### Returns

- **mode** (*int*) The octal permissions for the file
- size (int) The size of the file
- **mtime** (*int*) The last modified time for the file

**streaming\_shell** (*command*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*, *decode=True*) Send an ADB shell command to the device, yielding each line of output.

#### **Parameters**

- **command** (str) The shell command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManager.read()
- **decode** (bool) Whether to decode the output to utf8 before returning

**Yields** bytes, str – The line-by-line output of the ADB shell command as a string if decode is True, otherwise as bytes.

class adb\_shell.adb\_device.AdbDeviceTcp (host,

port=5555,

fault\_transport\_timeout\_s=None, banner=None)

de-

Bases: adb shell.adb device.AdbDevice

A class with methods for connecting to a device via TCP and executing ADB commands.

#### **Parameters**

- host (str) The address of the device; may be an IP address or a host name
- port (int) The device port to which we are connecting (default is 5555)
- default\_transport\_timeout\_s (float, None) Default timeout in seconds for TCP packets, or None
- banner (str, bytes, None) The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via socket. gethostname()

## \_available

Whether an ADB connection to the device has been established

Type bool

#### banner

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

## \_default\_transport\_timeout\_s

Default timeout in seconds for TCP packets, or None

Type float, None

## \_local\_id

The local ID that is used for ADB transactions; the value is incremented each time and is always in the range  $[1, 2^32]$ 

Type int

### \_maxdata

Maximum amount of data in an ADB packet

Type int

## \_transport

The transport that is used to connect to the device

**Type** TcpTransport

class adb\_shell.adb\_device.AdbDeviceUsb (serial=None,

port\_path=None,

fault\_transport\_timeout\_s=None, banner=None)

de-

Bases: adb shell.adb device.AdbDevice

A class with methods for connecting to a device via USB and executing ADB commands.

#### **Parameters**

- serial (str, None) The USB device serial ID
- port\_path (TODO, None) TODO
- default\_transport\_timeout\_s (float, None) Default timeout in seconds for USB packets, or None

• banner (str, bytes, None) - The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via socket. gethostname()

Raises adb\_shell.exceptions.InvalidTransportError - Raised if package was not installed with the "usb" extras option (pip install adb-shell[usb])

#### available

Whether an ADB connection to the device has been established

Type bool

#### banner

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

## \_default\_transport\_timeout\_s

Default timeout in seconds for USB packets, or None

Type float, None

#### \_local\_id

The local ID that is used for ADB transactions; the value is incremented each time and is always in the range  $[1, 2^32)$ 

Type int

#### \_maxdata

Maximum amount of data in an ADB packet

Type int

## \_transport

The transport that is used to connect to the device

Type UsbTransport

class adb\_shell.adb\_device.\_AdbIOManager(transport)

Bases: object

A class for handling all ADB I/O.

## **Notes**

When the self.\_store\_lock and self.\_transport\_lock locks are held at the same time, it must always be the case that the self.\_transport\_lock is acquired first. This ensures that there is no potential for deadlock.

**Parameters transport** (BaseTransport) – A transport for communicating with the device; must be an instance of a subclass of BaseTransport

## \_packet\_store

A store for holding packets that correspond to different ADB streams

**Type** \_*AdbPacketStore* 

## \_store\_lock

A lock for protecting self.\_packet\_store (this lock is never held for long)

Type Lock

## \_transport

A transport for communicating with the device; must be an instance of a subclass of BaseTransport

#### **Type** BaseTransport

#### \_transport\_lock

A lock for protecting self.\_transport

Type Lock

## \_read\_bytes\_from\_device (length, adb\_info)

Read length bytes from the device.

#### **Parameters**

- length (int) We will read packets until we get this length of data
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

**Returns** The data that was read

Return type bytes

Raises adb\_shell.exceptions.AdbTimeoutError - Did not read length bytes in time

## \_read\_expected\_packet\_from\_device (expected\_cmds, adb\_info)

Read packets from the device until we get an expected packet type.

#### **Parameters**

- expected\_cmds (list[bytes]) We will read packets until we encounter one whose "command" field is in expected\_cmds
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### **Returns**

- cmd (bytes) The received command, which is in adb\_shell.constants. WIRE\_TO\_ID and must be in expected\_cmds
- **arg0** (*int*) TODO
- arg1 (int) TODO
- data (bytes) The data that was read

Raises adb\_shell.exceptions.AdbTimeoutError - Never got one of the expected responses

## $\_{\tt read\_packet\_from\_device} (adb\_info)$

Read a complete ADB packet (header + data) from the device.

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

## Returns

- cmd (bytes) The received command, which is in adb\_shell.constants. WIRE\_TO\_ID and must be in expected\_cmds
- **arg0** (*int*) TODO
- **arg1** (*int*) TODO
- bytes The data that was read

## Raises

• adb shell.exceptions.InvalidCommandError - Unknown command

 adb\_shell.exceptions.InvalidChecksumError – Received checksum does not match the expected checksum

## \_send(msg, adb\_info)

Send a message to the device.

- 1. Send the message header (adb\_shell.adb\_message.AdbMessage.pack)
- 2. Send the message data

## **Parameters**

- msg (AdbMessage) The data that will be sent
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### close()

Close the connection via the provided transport's close () method and clear the packet store.

connect (banner, rsa\_keys, auth\_timeout\_s, auth\_callback, adb\_info)

Establish an ADB connection to the device.

- 1. Use the transport to establish a connection
- 2. Send a b 'CNXN' message
- 3. Read the response from the device
- 4. If cmd is not b'AUTH', then authentication is not necesary and so we are done
- 5. If no rsa\_keys are provided, raise an exception
- 6. Loop through our keys, signing the last banner2 that we received
  - 1. If the last arg0 was not adb\_shell.constants.AUTH\_TOKEN, raise an exception
  - 2. Sign the last banner2 and send it in an b'AUTH' message
  - 3. Read the response from the device
  - 4. If cmd is b'CNXN', we are done
- 7. None of the keys worked, so send rsa\_keys[0]'s public key; if the response does not time out, we must have connected successfully

## **Parameters**

- banner (bytearray, bytes) The hostname of the machine where the Python interpreter is currently running (adb\_shell.adb\_device.AdbDevice.\_banner)
- rsa\_keys (list, None) A list of signers of type CryptographySigner, PycryptodomeAuthSigner, or PythonRSASigner
- auth\_timeout\_s (float, None) The time in seconds to wait for a b'CNXN' authentication response
- auth\_callback (function, None) Function callback invoked when the connection needs to be accepted on the device
- adb\_info (\_AdbTransactionInfo) Info and settings for this connection attempt

## Returns

- bool Whether the connection was established
- maxdata (int) Maximum amount of data in an ADB packet

#### Raises

- adb\_shell.exceptions.DeviceAuthError Device authentication required, no keys available
- adb\_shell.exceptions.InvalidResponseError Invalid auth response from the device

read (expected\_cmds, adb\_info, allow\_zeros=False)

Read packets from the device until we get an expected packet type.

- 1. See if the expected packet is in the packet store
- 2. While the time limit has not been exceeded:
  - 1. See if the expected packet is in the packet store
  - 2. Read a packet from the device. If it matches what we are looking for, we are done. If it corresponds to a different stream, add it to the store.
- 3. Raise a timeout exception

#### **Parameters**

- **expected\_cmds** (*list[bytes]*) We will read packets until we encounter one whose "command" field is in expected\_cmds
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- allow\_zeros (bool) Whether to allow the received arg0 and arg1 values to match with 0, in addition to adb\_info.remote\_id and adb\_info.local\_id, respectively

#### Returns

- cmd (bytes) The received command, which is in adb\_shell.constants. WIRE\_TO\_ID and must be in expected\_cmds
- arg0 (int) TODO
- arg1 (int) TODO
- data (bytes) The data that was read

Raises adb\_shell.exceptions.AdbTimeoutError - Never got one of the expected responses

send (msg, adb\_info)

Send a message to the device.

#### **Parameters**

- msg (AdbMessage) The data that will be sent
- $adb\_info$  (\_AdbTransactionInfo) Info and settings for this ADB transaction

## adb\_shell.adb\_device\_async module

Implement the AdbDeviceAsync class, which can connect to a device and run ADB shell commands.

- \_AdbIOManagerAsync
  - \_AdbIOManagerAsync.\_read\_bytes\_from\_device()

```
- _AdbIOManagerAsync._read_expected_packet_from_device()
   - _AdbIOManagerAsync._read_packet_from_device()
   - _AdbIOManagerAsync._send()
   - _AdbIOManagerAsync.close()
   - AdbIOManagerAsync.connect()
   - AdbIOManagerAsync.read()
   - _AdbIOManagerAsync.send()
• AdbDeviceAsync
   - AdbDeviceAsync. clse()
   - AdbDeviceAsync._filesync_flush()
   - AdbDeviceAsync._filesync_read()
   - AdbDeviceAsync._filesync_read_buffered()
   - AdbDeviceAsync._filesync_read_until()
   - AdbDeviceAsync._filesync_send()
   - AdbDeviceAsync._okay()
   - AdbDeviceAsync. open()
   - AdbDeviceAsync._pull()
   - AdbDeviceAsync._push()
   - AdbDeviceAsync._read_until()
   - AdbDeviceAsync. read until close()
   - AdbDeviceAsync._service()
   - AdbDeviceAsync._streaming_command()
   - AdbDeviceAsync._streaming_service()
   - AdbDeviceAsync.available
   - AdbDeviceAsync.close()
   - AdbDeviceAsync.connect()
   - AdbDeviceAsync.list()
   - AdbDeviceAsync.max chunk size
   - AdbDeviceAsync.pull()
   - AdbDeviceAsync.push()
   - AdbDeviceAsync.root()
   - AdbDeviceAsync.shell()
   - AdbDeviceAsync.stat()
   - AdbDeviceAsync.streaming_shell()
```

• AdbDeviceTcpAsync

Bases: object

A class with methods for connecting to a device and executing ADB commands.

#### **Parameters**

- **transport** (BaseTransportAsync) A user-provided transport for communicating with the device; must be an instance of a subclass of BaseTransportAsync
- default\_transport\_timeout\_s (float, None) Default timeout in seconds for transport packets, or None
- banner (str, bytes, None) The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via socket. gethostname()

Raises adb\_shell.exceptions.InvalidTransportError - The passed transport is not an instance of a subclass of BaseTransportAsync

#### available

Whether an ADB connection to the device has been established

Type bool

#### \_banner

The hostname of the machine where the Python interpreter is currently running

**Type** bytearray, bytes

## \_default\_transport\_timeout\_s

Default timeout in seconds for transport packets, or None

Type float, None

## \_io\_manager

Used for handling all ADB I/O

**Type** \_AdbIOManagerAsync

## \_local\_id

The local ID that is used for ADB transactions; the value is incremented each time and is always in the range  $[1, 2^32)$ 

Type int

#### \_local\_id\_lock

A lock for protecting \_local\_id; this is never held for long

Type Lock

## \_maxdata

Maximum amount of data in an ADB packet

Type int

## \_transport

The transport that is used to connect to the device; must be a subclass of BaseTransportAsync

**Type** BaseTransportAsync

## async \_clse(adb\_info)

Send a b'CLSE' message and then read a b'CLSE' message.

Warning: This is not to be confused with the AdbDeviceAsync.close() method!

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

## async \_filesync\_flush (adb\_info, filesync\_info)

Write the data in the buffer up to filesync\_info.send\_idx, then set filesync\_info.send idx to 0.

#### **Parameters**

- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- **filesync\_info** (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

## async \_filesync\_read (expected\_ids, adb\_info, filesync\_info)

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
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

#### Returns

- command\_id (bytes) The received header ID
- tuple The contents of the header
- data (bytearray, None) The received data, or None if the command ID is adb\_shell. constants.STAT

## Raises

- adb\_shell.exceptions.AdbCommandFailureException Command failed
- adb\_shell.exceptions.InvalidResponseError Received response was not in expected ids

## async \_filesync\_read\_buffered (size, adb\_info, filesync\_info)

Read size bytes of data from self.recv\_buffer.

## **Parameters**

- size (int) The amount of data to read
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

## Returns result – The read data

Return type bytearray

## \_filesync\_read\_until (expected\_ids, finish\_ids, adb\_info, filesync\_info)

Useful wrapper around AdbDeviceAsync.\_filesync\_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
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

#### **Yields**

- **cmd\_id** (*bytes*) The received header ID
- header (tuple) TODO
- data (bytearray) The received data

**async** \_filesync\_send (command\_id, adb\_info, filesync\_info, data=b", size=None) 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.
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- **filesync\_info** (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction
- data (str, bytes) Optional data to send, must set data or size.
- **size** (*int*, *None*) Optionally override size from len(data).

## \_get\_transport\_timeout\_s (transport\_timeout\_s)

Use the provided transport\_timeout\_s if it is not None; otherwise, use self. \_default\_transport\_timeout\_s

Parameters transport\_timeout\_s (float, None) - The potential transport timeout

**Returns** transport\_timeout\_s if it is not None; otherwise, self. \_default\_transport\_timeout\_s

## Return type float

#### async \_okay (adb\_info)

Send an b'OKAY' mesage.

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

async \_open (destination, transport\_timeout\_s, read\_timeout\_s, timeout\_s)

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

- 1. send() an b'OPEN' command to the device that specifies the local\_id
- 2. read() the response from the device and fill in the adb\_info.remote\_id attribute

#### **Parameters**

destination (bytes) - b'SERVICE:COMMAND'

- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish

Returns adb\_info - Info and settings for this ADB transaction

**Return type** \_AdbTransactionInfo

**async** \_pull (device\_path, stream, progress\_callback, adb\_info, filesync\_info)

Pull a file from the device into the file-like local\_path.

#### **Parameters**

- **device\_path** (str) The file on the device that will be pulled
- stream (AsyncBufferedIOBase) File-like object for writing to
- progress\_callback (function, None) Callback method that accepts device\_path, bytes\_written, and total\_bytes
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- filesync\_info (\_FileSyncTransactionInfo) Data and storage for this FileSync transaction

**async** \_push (stream, device\_path, st\_mode, mtime, progress\_callback, adb\_info, filesync\_info) Push a file-like object to the device.

## **Parameters**

- stream (AsyncBufferedReader) File-like object for reading from
- **device\_path** (str) Destination on the device to write to
- st\_mode (int) Stat mode for the file
- mtime (int) Modification time
- progress\_callback (function, None) Callback method that accepts device\_path, bytes\_written, and total\_bytes
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

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

async \_read\_until (expected\_cmds, adb\_info)

Read a packet, acknowledging any write packets.

- 1. Read data via \_AdbIOManagerAsync.read()
- 2. If a b'WRTE' packet is received, send an b'OKAY' packet via AdbDeviceAsync.\_okay()
- 3. Return the cmd and data that were read by \_AdbIOManagerAsync.read()

## **Parameters**

- expected\_cmds (list[bytes]) \_AdbIOManagerAsync.read() will look for a packet whose command is in expected\_cmds
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### Returns

- cmd (bytes) The command that was received by \_AdbIOManagerAsync. read(), which is in adb\_shell.constants.WIRE\_TO\_ID and must be in expected\_cmds
- data (bytes) The data that was received by \_AdbIOManagerAsync.read()

#### \_read\_until\_close(adb\_info)

Yield packets until a b 'CLSE' packet is received.

- Read the cmd and data fields from a b'CLSE' or b'WRTE' packet via AdbDeviceAsync. \_read\_until()
- 2. If cmd is b'CLSE', then send a b'CLSE' message and stop
- 3. Yield data and repeat

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

**Yields data** (bytes) – The data that was read by AdbDeviceAsync.\_read\_until()

#### **Parameters**

- **service** (bytes) The ADB service to talk to (e.g., b'shell')
- **command** (bytes) The command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **decode** (bool) Whether to decode the output to utf8 before returning

Returns The output of the ADB command as a string if decode is True, otherwise as bytes.

Return type bytes, str

- \_streaming\_command(service, command, transport\_timeout\_s, read\_timeout\_s, timeout\_s)
  One complete set of packets for a single command.
  - 1. \_open() a new connection to the device, where the destination parameter is service:command
  - 2. Read the response data via AdbDeviceAsync.\_read\_until\_close()

**Note:** All the data is held in memory, and thus large responses will be slow and can fill up memory.

#### **Parameters**

- **service** (bytes) The ADB service (e.g., b'shell', as used by AdbDeviceAsync.shell())
- command (bytes) The service command
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish

**Yields** bytes – The responses from the service.

\_streaming\_service (service, command, transport\_timeout\_s=None, read\_timeout\_s=10.0, decode=True)
Send an ADB command to the device, yielding each line of output.

#### **Parameters**

- **service** (bytes) The ADB service to talk to (e.g., b'shell')
- command (bytes) The command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- **decode** (bool) Whether to decode the output to utf8 before returning

**Yields** *bytes*, *str* – The line-by-line output of the ADB command as a string if decode is True, otherwise as bytes.

#### property available

Whether or not an ADB connection to the device has been established.

Returns self.\_available

Return type bool

## async close()

Close the connection via the provided transport's close () method.

async connect (rsa\_keys=None, transport\_timeout\_s=None, auth\_timeout\_s=10.0, read\_timeout\_s=10.0, auth\_callback=None)

Establish an ADB connection to the device.

See \_AdbIOManagerAsync.connect().

#### **Parameters**

- rsa\_keys (list, None) A list of signers of type CryptographySigner, PycryptodomeAuthSigner, or PythonRSASigner
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk\_write()

- auth\_timeout\_s (float, None) The time in seconds to wait for a b'CNXN' authentication response
- read\_timeout\_s (float) The total time in seconds to wait for expected commands in \_AdbIOManagerAsync.\_read\_expected\_packet\_from\_device()
- auth\_callback (function, None) Function callback invoked when the connection needs to be accepted on the device

**Returns** Whether the connection was established (AdbDeviceAsync.available)

Return type bool

**async exec\_out** (command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True)

Send an ADB exec-out command to the device.

https://www.linux-magazine.com/Issues/2017/195/Ask-Klaus

#### **Parameters**

- command (str) The exec-out command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read() and BaseTransport.bulk write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **decode** (bool) Whether to decode the output to utf8 before returning

**Returns** The output of the ADB exec-out command as a string if decode is True, otherwise as bytes.

Return type bytes, str

**async list** (*device\_path*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*) Return a directory listing of the given path.

## **Parameters**

- **device\_path** (str) Directory to list.
- transport\_timeout\_s (float, None) Expected timeout for any part of the pull.
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()

Returns files - Filename, mode, size, and mtime info for the files in the directory

**Return type** list[*DeviceFile*]

### property max\_chunk\_size

Maximum chunk size for filesync operations

**Returns** Minimum value based on adb\_shell.constants.MAX\_CHUNK\_SIZE and \_max\_data / 2, fallback to legacy adb\_shell.constants.MAX\_PUSH\_DATA

Return type int

async pull (device\_path, local\_path, progress\_callback=None, transport\_timeout\_s=None, read\_timeout\_s=10.0)

Pull a file from the device.

#### **Parameters**

- **device\_path** (str) The file on the device that will be pulled
- **local\_path** (*str*) The path to where the file will be downloaded
- progress\_callback (function, None) Callback method that accepts device path, bytes written, and total bytes
- transport\_timeout\_s (float, None) Expected timeout for any part of the pull.
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- **async push** (local\_path, device\_path, st\_mode=33272, mtime=0, progress\_callback=None, trans-port\_timeout\_s=None, read\_timeout\_s=10.0)

  Push a file or directory to the device.

#### **Parameters**

- local\_path (str) Either a filename or a directory to push to the device
- **device\_path** (str) Destination on the device to write to
- st mode (int) Stat mode for local path
- **mtime** (*int*) Modification time to set on the file
- progress\_callback (function, None) Callback method that accepts device path, bytes written, and total bytes
- transport\_timeout\_s (float, None) Expected timeout for any part of the push
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- **async root** (*transport\_timeout\_s=None*, *read\_timeout\_s=10.0*, *timeout\_s=None*) Gain root access.

The device must be rooted in order for this to work.

## **Parameters**

- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- async shell(command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True)

Send an ADB shell command to the device.

## **Parameters**

• **command** (str) – The shell command that will be sent

- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish
- **decode** (bool) Whether to decode the output to utf8 before returning

**Returns** The output of the ADB shell command as a string if decode is True, otherwise as bytes.

## Return type bytes, str

**async stat** (*device\_path*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*) Get a file's stat() information.

#### **Parameters**

- **device\_path** (str) The file on the device for which we will get information.
- transport\_timeout\_s (float, None) Expected timeout for any part of the pull.
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()

#### Returns

- mode (int) The octal permissions for the file
- size (int) The size of the file
- **mtime** (*int*) The last modified time for the file

**streaming\_shell** (*command*, *transport\_timeout\_s=None*, *read\_timeout\_s=10.0*, *decode=True*) Send an ADB shell command to the device, yielding each line of output.

## **Parameters**

- **command** (str) The shell command that will be sent
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransportAsync.bulk\_read() and BaseTransportAsync.bulk\_write()
- read\_timeout\_s (float) The total time in seconds to wait for a b'CLSE' or b'OKAY' command in \_AdbIOManagerAsync.read()
- **decode** (bool) Whether to decode the output to utf8 before returning

**Yields** bytes, str – The line-by-line output of the ADB shell command as a string if decode is True, otherwise as bytes.

Bases: adb\_shell.adb\_device\_async.AdbDeviceAsync

A class with methods for connecting to a device via TCP and executing ADB commands.

#### **Parameters**

• host (str) – The address of the device; may be an IP address or a host name

- port (int) The device port to which we are connecting (default is 5555)
- default\_transport\_timeout\_s (float, None) Default timeout in seconds for TCP packets, or None
- banner (str, bytes, None) The hostname of the machine where the Python interpreter is currently running; if it is not provided, it will be determined via socket. gethostname()

#### available

Whether an ADB connection to the device has been established

Type bool

#### banner

The hostname of the machine where the Python interpreter is currently running

Type bytearray, bytes

## \_default\_transport\_timeout\_s

Default timeout in seconds for TCP packets, or None

Type float, None

## \_local\_id

The local ID that is used for ADB transactions; the value is incremented each time and is always in the range  $[1, 2^32)$ 

Type int

#### maxdata

Maximum amount of data in an ADB packet

Type int

## \_transport

The transport that is used to connect to the device

**Type** *TcpTransportAsync* 

class adb\_shell.adb\_device\_async.\_AdbIOManagerAsync(transport)

Bases: object

A class for handling all ADB I/O.

## **Notes**

When the self.\_store\_lock and self.\_transport\_lock locks are held at the same time, it must always be the case that the self.\_transport\_lock is acquired first. This ensures that there is no potential for deadlock.

Parameters transport (BaseTransportAsync) - A transport for communicating with the device; must be an instance of a subclass of BaseTransportAsync

#### packet store

A store for holding packets that correspond to different ADB streams

Type \_AdbPacketStore

## store lock

A lock for protecting self.\_packet\_store (this lock is never held for long)

Type Lock

#### transport

A transport for communicating with the device; must be an instance of a subclass of BaseTransportAsync

**Type** BaseTransportAsync

## \_transport\_lock

A lock for protecting self.\_transport

Type Lock

## async \_read\_bytes\_from\_device(length, adb\_info)

Read length bytes from the device.

#### **Parameters**

- length (int) We will read packets until we get this length of data
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

Returns The data that was read

Return type bytes

Raises adb\_shell.exceptions.AdbTimeoutError - Did not read length bytes in time

## async \_read\_expected\_packet\_from\_device (expected\_cmds, adb\_info)

Read packets from the device until we get an expected packet type.

#### **Parameters**

- **expected\_cmds** (*list[bytes]*) We will read packets until we encounter one whose "command" field is in expected\_cmds
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### Returns

- cmd (bytes) The received command, which is in adb\_shell.constants.

  WIRE\_TO\_ID and must be in expected\_cmds
- **arg0** (*int*) TODO
- arg1 (int) TODO
- data (bytes) The data that was read

Raises adb\_shell.exceptions.AdbTimeoutError - Never got one of the expected responses

## $\verb"async _read_packet_from_device" (adb_info)$

Read a complete ADB packet (header + data) from the device.

Parameters adb\_info (\_AdbTransactionInfo) - Info and settings for this ADB transaction

#### **Returns**

- cmd (bytes) The received command, which is in adb\_shell.constants.

  WIRE\_TO\_ID and must be in expected\_cmds
- arg0 (int) TODO
- arg1 (int) TODO
- bytes The data that was read

#### Raises

- adb\_shell.exceptions.InvalidCommandError Unknown command
- adb\_shell.exceptions.InvalidChecksumError Received checksum does not match the expected checksum

#### async \_send(msg, adb\_info)

Send a message to the device.

- 1. Send the message header (adb\_shell.adb\_message.AdbMessage.pack)
- 2. Send the message data

#### **Parameters**

- msg (AdbMessage) The data that will be sent
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### async close()

Close the connection via the provided transport's close () method and clear the packet store.

async connect (banner, rsa\_keys, auth\_timeout\_s, auth\_callback, adb\_info)

Establish an ADB connection to the device.

- 1. Use the transport to establish a connection
- 2. Send a b 'CNXN' message
- 3. Read the response from the device
- 4. If cmd is not b'AUTH', then authentication is not necesary and so we are done
- 5. If no rsa\_keys are provided, raise an exception
- 6. Loop through our keys, signing the last banner2 that we received
  - 1. If the last arg0 was not adb\_shell.constants.AUTH\_TOKEN, raise an exception
  - 2. Sign the last banner2 and send it in an b'AUTH' message
  - 3. Read the response from the device
  - 4. If cmd is b'CNXN', we are done
- 7. None of the keys worked, so send rsa\_keys[0]'s public key; if the response does not time out, we must have connected successfully

#### **Parameters**

- banner (bytearray, bytes) The hostname of the machine where the Python interpreter is currently running (adb\_shell.adb\_device.AdbDevice.\_banner)
- rsa\_keys (list, None) A list of signers of type CryptographySigner, PycryptodomeAuthSigner, or PythonRSASigner
- auth\_timeout\_s (float, None) The time in seconds to wait for a b'CNXN' authentication response
- auth\_callback (function, None) Function callback invoked when the connection needs to be accepted on the device
- adb\_info (\_AdbTransactionInfo) Info and settings for this connection attempt

#### Returns

- bool Whether the connection was established
- maxdata (int) Maximum amount of data in an ADB packet

#### Raises

- adb\_shell.exceptions.DeviceAuthError Device authentication required, no keys available
- adb\_shell.exceptions.InvalidResponseError Invalid auth response from the device

async read(expected\_cmds, adb\_info, allow\_zeros=False)

Read packets from the device until we get an expected packet type.

- 1. See if the expected packet is in the packet store
- 2. While the time limit has not been exceeded:
  - 1. See if the expected packet is in the packet store
  - 2. Read a packet from the device. If it matches what we are looking for, we are done. If it corresponds to a different stream, add it to the store.
- 3. Raise a timeout exception

#### **Parameters**

- **expected\_cmds** (*list[bytes]*) We will read packets until we encounter one whose "command" field is in expected\_cmds
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction
- allow\_zeros (bool) Whether to allow the received arg0 and arg1 values to match with 0, in addition to adb\_info.remote\_id and adb\_info.local\_id, respectively

#### **Returns**

- cmd (bytes) The received command, which is in adb\_shell.constants. WIRE\_TO\_ID and must be in expected\_cmds
- **arg0** (*int*) TODO
- **arg1** (*int*) TODO
- data (bytes) The data that was read

Raises adb\_shell.exceptions.AdbTimeoutError - Never got one of the expected responses

async send(msg, adb\_info)

Send a message to the device.

#### **Parameters**

- msg (AdbMessage) The data that will be sent
- adb\_info (\_AdbTransactionInfo) Info and settings for this ADB transaction

#### adb shell.adb message module

Functions and an AdbMessage class for packing and unpacking ADB messages.

## **Contents**

```
• AdbMessage
       - AdbMessage.checksum
       - AdbMessage.pack()
   • checksum()
   • int_to_cmd()
   • unpack()
class adb shell.adb message. AdbMessage (command, arg0, arg1, data=b")
    Bases: object
    A helper class for packing ADB messages.
         Parameters
              • command (bytes) - A command; examples used in this package include adb shell.
                constants.AUTH, adb_shell.constants.CNXN, adb_shell.constants.
               CLSE, adb shell.constants.OPEN, and adb shell.constants.OKAY
              • arg0 (int) - Usually the local ID, but connect() and connect()
                          adb_shell.constants.VERSION,
                                                              adb_shell.constants.
               AUTH_SIGNATURE, and adb_shell.constants.AUTH_RSAPUBLICKEY
              • arg1 (int) - Usually the remote ID, but connect() and connect() provide
                adb shell.constants.MAX ADB DATA
              • data (bytes) - The data that will be sent
    arg0
         Usually the local ID, but connect() and connect() provide adb shell.constants.
         VERSION, adb_shell.constants.AUTH_SIGNATURE, and adb_shell.constants.
         AUTH RSAPUBLICKEY
            Type int
    arg1
         Usually the remote ID, but connect() and connect() provide adb_shell.constants.
         MAX ADB DATA
            Type int
    command
         The input parameter command converted to an integer via adb_shell.constants.ID_TO_WIRE
            Type int
    data
         The data that will be sent
            Type bytes
    magic
         self.command with its bits flipped; in other words, self.command + self.magic == 2**32
         - 1
            Type int
```

property checksum

Return checksum (self.data)

```
Returns The checksum of self.data
             Return type int
     pack()
          Returns this message in an over-the-wire format.
             Returns The message packed into the format required by ADB
             Return type bytes
adb_shell.adb_message.checksum(data)
     Calculate the checksum of the provided data.
          Parameters data (bytearray, bytes, str) - The data
          Returns The checksum
          Return type int
adb_shell.adb_message.int_to_cmd(n)
     Convert from an integer (4 bytes) to an ADB command.
          Parameters n (int) – The integer that will be converted to an ADB command
          Returns The ADB command (e.g., 'CNXN')
          Return type str
adb shell.adb message.unpack(message)
     Unpack a received ADB message.
          Parameters message (bytes) – The received message
          Returns
               • cmd (int) – The ADB command
               • arg0 (int) - TODO
               • arg1 (int) – TODO
               • data_length (int) – The length of the message's data
               • data_checksum (int) – The checksum of the message's data
          Raises ValueError – Unable to unpack the ADB command.
adb shell.constants module
Constants used throughout the code.
adb_shell.constants.AUTH_RSAPUBLICKEY = 3
     AUTH constant for arg0
adb shell.constants.AUTH SIGNATURE = 2
     AUTH constant for arg0
adb_shell.constants.AUTH_TOKEN = 1
     AUTH constant for arg0
```

adb shell.constants.CLASS = 255

From adb.h

```
adb shell.constants.DEFAULT AUTH TIMEOUT S = 10.0
    Default authentication timeout (in s) for adb_shell.adb_device.AdbDevice.connect() and
    adb_shell.adb_device_async.AdbDeviceAsync.connect()
adb_shell.constants.DEFAULT_PUSH_MODE = 33272
    Default mode for pushed files.
adb_shell.constants.DEFAULT_READ_TIMEOUT_S = 10.0
    Default total timeout (in s) for reading data from the device
adb_shell.constants.FILESYNC_IDS = (b'DATA', b'DENT', b'DONE', b'FAIL', b'LIST', b'OKAY', l
    Commands that are recognized by adb_shell.adb_device.AdbDevice._filesync_read() and
    adb_shell.adb_device_async.AdbDeviceAsync._filesync_read()
adb_shell.constants.FILESYNC_ID_TO_WIRE = {b'DATA': 1096040772, b'DENT': 1414415684, b'DON
    A dictionary where the keys are the commands in FILESYNC_IDS and the values are the keys converted to
    integers
adb_shell.constants.FILESYNC_LIST_FORMAT = b'<5I'</pre>
    The format for FileSync "list" messages
adb_shell.constants.FILESYNC_PULL_FORMAT = b'<2I'</pre>
    The format for FileSync "pull" messages
adb_shell.constants.FILESYNC_PUSH_FORMAT = b'<2I'</pre>
    The format for FileSync "push" messages
adb shell.constants.FILESYNC STAT FORMAT = b'<41'
    The format for FileSync "stat" messages
adb_shell.constants.FILESYNC_WIRE_TO_ID = {1096040772: b'DATA', 1145980243: b'SEND', 1165
    A dictionary where the keys are integers and the values are their corresponding commands (type = bytes) from
    FILESYNC_IDS
adb_shell.constants.IDS = (b'AUTH', b'CLSE', b'CNXN', b'OKAY', b'OPEN', b'SYNC', b'WRTE')
    Commands
                               recognized
                                            by
                                                 adb_shell.adb_device._AdbIOManager.
    _read_packet_from_device()
                                             and
                                                           adb_shell.adb_device_async.
    _AdbIOManagerAsync._read_packet_from_device()
adb_shell.constants.ID_TO_WIRE = {b'AUTH': 1213486401, b'CLSE': 1163086915, b'CNXN': 13144
    A dictionary where the keys are the commands in IDS and the values are the keys converted to integers
adb shell.constants.MAX ADB DATA = 1048576
    //android.googlesource.com/platform/system/core/+/master/adb/adb.h
         Type Maximum amount of data in an ADB packet. According to
         Type https
adb_shell.constants.MAX_CHUNK_SIZE = 65536
    //android.googlesource.com/platform/system/core/+/master/adb/SYNC.TXT
         Type Maximum chunk size. According to https
```

1.1. adb\_shell package

The size of an ADB message

adb\_shell.constants.MAX\_PUSH\_DATA = 2048

Maximum size of a filesync DATA packet. Default size.

adb\_shell.constants.MESSAGE\_FORMAT = b'<6I'
An ADB message is 6 words in little-endian.

adb\_shell.constants.MESSAGE\_SIZE = 24

```
adb shell.constants.PROTOCOL = 1
    From adb.h
adb shell.constants.SUBCLASS = 66
    From adb.h
adb shell.constants.VERSION = 16777216
    ADB protocol version.
adb_shell.constants.WIRE_TO_ID = {1129208147: b'SYNC', 1163086915: b'CLSE', 1163154007:
    A dictionary where the keys are integers and the values are their corresponding commands (type = bytes) from
adb shell.exceptions module
ADB-related exceptions.
exception adb_shell.exceptions.AdbCommandFailureException
    Bases: Exception
    Ab'FAIL' packet was received.
exception adb_shell.exceptions.AdbConnectionError
    Bases: Exception
    ADB command not sent because a connection to the device has not been established.
exception adb_shell.exceptions.AdbTimeoutError
    Bases: Exception
    ADB command did not complete within the specified time.
exception adb_shell.exceptions.DeviceAuthError(message, *args)
    Bases: Exception
    Device authentication failed.
exception adb_shell.exceptions.DevicePathInvalidError
    Bases: Exception
    A file command was passed an invalid path.
exception adb_shell.exceptions.InvalidChecksumError
    Bases: Exception
    Checksum of data didn't match expected checksum.
exception adb_shell.exceptions.InvalidCommandError
    Bases: Exception
    Got an invalid command.
exception adb shell.exceptions.InvalidResponseError
    Bases: Exception
    Got an invalid response to our command.
exception adb_shell.exceptions.InvalidTransportError
    Bases: Exception
    The provided transport does not implement the necessary methods: close, connect, bulk_read, and
    bulk_write.
```

```
exception adb_shell.exceptions.PushFailedError
    Bases: Exception
    Pushing a file failed for some reason.
exception adb_shell.exceptions.TcpTimeoutException
    Bases: Exception
    TCP connection timed read/write operation exceeded the allowed time.
exception adb_shell.exceptions.UsbDeviceNotFoundError
    Bases: Exception
    TODO
exception adb_shell.exceptions.UsbReadFailedError(msg, usb_error)
    Bases: Exception
    TODO
         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_shell.exceptions.UsbWriteFailedError
    Bases: Exception
    adb_shell.transport.usb_transport.UsbTransport.bulk_write() failed.
```

## adb\_shell.hidden\_helpers module

Implement helpers for the AdbDevice and AdbDeviceAsync classes.

## **Contents**

```
    _AdbPacketStore
    _AdbPacketStore.__contains__()
    _AdbPacketStore.__len__()
    _AdbPacketStore.clear()
    _AdbPacketStore.clear_all()
    _AdbPacketStore.find()
    _AdbPacketStore.find_allow_zeros()
    _AdbPacketStore.get()
    _AdbPacketStore.put()
    _AdbTransactionInfo
    _AdbTransactionInfo.args_match()
    _FileSyncTransactionInfo
```

```
- FileSyncTransactionInfo.can add to send buffer()
   • get banner()
   • get_files_to_push()
class adb_shell.hidden_helpers.DeviceFile (filename, mode, size, mtime)
    Bases: tuple
    asdict()
         Return a new OrderedDict which maps field names to their values.
    _field_defaults = {}
    fields = ('filename', 'mode', 'size', 'mtime')
    _fields_defaults = {}
    classmethod _make(iterable)
         Make a new DeviceFile object from a sequence or iterable
    _replace(**kwds)
         Return a new DeviceFile object replacing specified fields with new values
    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_shell.hidden_helpers._AdbPacketStore
    Bases: object
    A class for storing ADB packets.
    This class is used to support multiple streams.
     dict
         Α
            dictionary of dictionaries of queues.
                                                      The first (outer) dictionary keys are
              arg1
                      return
                              values
                                      from the
                                                  adb_shell.adb_device._AdbIOManager.
                                                            adb_shell.adb_device_async.
         _read_packet_from_device()
                                                and
         _AdbIOManagerAsync._read_packet_from_device() methods.
                                                                        The second (inner)
         dictionary keys are the argo return values from those methods. And the values of this inner dictionary
         are queues of (cmd, data) tuples.
            Type dict[int: dict[int: Queue]]
    clear (arg0, arg1)
         Delete the entry for (arg0, arg1), if it exists.
             Parameters
                        (int) - The arg0 return
                                                        value from
                                                                       the adb shell.
                  adb_device._AdbIOManager._read_packet_from_device()
                                  adb_shell.adb_device_async._AdbIOManagerAsync.
                  _read_packet_from_device() methods
                • arg1 (int) - The arg1 return
                                                        value from
                                                                       the adb shell.
                  adb device. AdbIOManager. read packet from device()
```

#### clear all()

Clear all the entries.

## find(arg0, arg1)

Find the entry corresponding to arg0 and arg1.

#### **Parameters**

- arg0 (int, None) The arg0 value that we are looking for; None serves as a wild-card
- arg1 (int, None) The arg1 value that we are looking for; None serves as a wild-card

**Returns** The (arg0, arg1) pair that was found in the dictionary of dictionaries, or None if no match was found

Return type tuple[int, int], None

## find\_allow\_zeros (arg0, arg1)

Find the entry corresponding to (arg0 or 0) and (arg1 or 0).

#### **Parameters**

- arg0 (int, None) The arg0 value that we are looking for; None serves as a wild-
- arg1 (int, None) The arg1 value that we are looking for; None serves as a wild-card

**Returns** The first matching (arg0, arg1) pair that was found in the dictionary of dictionaries, or None if no match was found

Return type tuple[int, int], None

```
get (arg0, arg1)
```

Get the next entry from the queue for arg0 and arg1.

This function assumes you have already checked that (arg0, arg1) in self.

## **Parameters**

- arg0 (int, None) The arg0 return value from the adb\_shell. adb\_device.\_AdbIOManager.\_read\_packet\_from\_device() and adb\_shell.adb\_device\_async.\_AdbIOManagerAsync. read packet from device() methods; None serves as a wildcard
- arg1 (int, None) The arg1 return value from the adb\_shell. adb\_device.\_AdbIOManager.\_read\_packet\_from\_device() and adb\_shell.adb\_device\_async.\_AdbIOManagerAsync. \_read\_packet\_from\_device() methods; None serves as a wildcard

#### **Returns**

- cmd (bytes) The ADB packet's command
- arg0 (int) The arg0 value from the returned packet
- arg1 (int) The arg1 value from the returned packet
- data (bytes) The ADB packet's data

```
put (arg0, arg1, cmd, data)
```

Add an entry to the queue for arg0 and arg1.

Note that a new dictionary entry will not be created if cmd == constants.CLSE.

#### **Parameters**

- arg0 (int) The arg0 return value from the adb\_shell. adb\_device.\_AdbIOManager.\_read\_packet\_from\_device()
   and adb\_shell.adb\_device\_async.\_AdbIOManagerAsync.\_read\_packet\_from\_device() methods
- arg1 (int) The arg1 return value from the adb\_shell. adb\_device.\_AdbIOManager.\_read\_packet\_from\_device() and adb\_shell.adb\_device\_async.\_AdbIOManagerAsync. \_read\_packet\_from\_device() methods
- cmd (bytes) The ADB packet's command
- data (bytes) The ADB packet's data

```
class adb_shell.hidden_helpers._AdbTransactionInfo(local\_id, remote\_id, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None)
```

Bases: object

A class for storing info and settings used during a single ADB "transaction."

Note that if timeout\_s is not None, then:

```
self.transport_timeout_s <= self.read_timeout_s <= self.timeout_s
```

If timeout\_s is None, the first inequality still applies.

#### **Parameters**

- local id (int) The ID for the sender (i.e., the device running this code)
- remote\_id (int) The ID for the recipient
- transport\_timeout\_s (float, None) Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read(),
  BaseTransport.bulk\_write(), BaseTransportAsync.bulk\_read(),
  and BaseTransportAsync.bulk\_write()
- **read\_timeout\_s** (*float*) The total time in seconds to wait for data and packets from the device
- timeout\_s (float, None) The total time in seconds to wait for the ADB command to finish

## local id

The ID for the sender (i.e., the device running this code)

Type int

#### read\_timeout\_s

The total time in seconds to wait for data and packets from the device

Type float

## remote\_id

The ID for the recipient

## Type int

#### timeout s

The total time in seconds to wait for the ADB command to finish

Type float, None

### transport\_timeout\_s

Timeout in seconds for sending and receiving data, or None; see BaseTransport.bulk\_read(), BaseTransport.bulk\_write(), BaseTransportAsync.bulk\_read(), and BaseTransportAsync.bulk\_write()

Type float, None

## args\_match (arg0, arg1, allow\_zeros=False)

Check if arg0 and arg1 match this object's remote\_id and local\_id attributes, respectively.

#### **Parameters**

- arg0 (int) The arg0 value from an ADB packet, which will be compared to this
  object's remote\_id attribute
- arg1 (int) The arg1 value from an ADB packet, which will be compared to this object's local\_id attribute
- allow\_zeros (bool) Whether to check if arg0 and arg1 match 0, in addition to this object's local\_id and remote\_id attributes

**Returns** Whether arg0 and arg1 match this object's local\_id and remote\_id attributes **Return type** bool

Bases: object

A class for storing info used during a single FileSync "transaction."

#### **Parameters**

- recv message format (bytes) The FileSync message format
- maxdata (int) Maximum amount of data in an ADB packet

## maxdata

Maximum amount of data in an ADB packet

Type int

#### recv buffer

A buffer for storing received data

**Type** bytearray

## ${\tt recv\_message\_format}$

The FileSync message format

Type bytes

## recv\_message\_size

The FileSync message size

Type int

#### send buffer

A buffer for storing data to be sent

## Type bytearray

#### send idx

The index in recv\_buffer that will be the start of the next data packet sent

Type int

## can\_add\_to\_send\_buffer(data\_len)

Determine whether data\_len bytes of data can be added to the send buffer without exceeding constants.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 constants.MAX\_ADB\_DATA

## Return type bool

```
adb_shell.hidden_helpers.get_banner()
```

Get the banner that will be signed in adb\_shell.adb\_device.AdbDevice.connect() / adb shell.adb device async.AdbDeviceAsync.connect().

Returns The hostname, or "unknown" if it could not be determined

## Return type bytearray

adb\_shell.hidden\_helpers.get\_files\_to\_push (local\_path, device\_path)

Get a list of the file(s) to push.

#### **Parameters**

- local\_path (str) A path to a local file or directory
- **device\_path** (str) A path to a file or directory on the device

#### Returns

- local\_path\_is\_dir (bool) Whether or not local\_path is a directory
- local\_paths (list[str]) A list of the file(s) to push
- device\_paths (list[str]) A list of destination paths on the device that corresponds to local\_paths

## 1.1.3 Module contents

ADB shell functionality.

This Python package implements ADB shell and FileSync functionality. It originated from python-adb.

**CHAPTER** 

**TWO** 

## **INSTALLATION**

pip install adb-shell

# 2.1 Async

To utilize the async version of this code, you must install into a Python 3.7+ environment via:

pip install adb-shell[async]

# 2.2 USB Support (Experimental)

To connect to a device via USB, install this package via:

pip install adb-shell[usb]

**CHAPTER** 

## **THREE**

## **EXAMPLE USAGE**

(Based on androidtv/adb\_manager.py)

```
from adb_shell.adb_device import AdbDeviceTcp, AdbDeviceUsb
from adb_shell.auth.sign_pythonrsa import PythonRSASigner
# Load the public and private keys
adbkey = 'path/to/adbkey'
with open(adbkey) as f:
   priv = f.read()
with open(adbkey + '.pub') as f:
    pub = f.read()
signer = PythonRSASigner(pub, priv)
# Connect
device1 = AdbDeviceTcp('192.168.0.222', 5555, default_transport_timeout_s=9.)
device1.connect(rsa_keys=[signer], auth_timeout_s=0.1)
# Connect via USB (package must be installed via `pip install adb-shell[usb])`
device2 = AdbDeviceUsb()
device2.connect(rsa_keys=[signer], auth_timeout_s=0.1)
# Send a shell command
response1 = device1.shell('echo TEST1')
response2 = device2.shell('echo TEST2')
```

## 3.1 Generate ADB Key Files

If you need to generate a key, you can do so as follows.

```
from adb_shell.auth.keygen import keygen
keygen('path/to/adbkey')
```

## **CHAPTER**

# **FOUR**

# **INDICES AND TABLES**

- genindex
- modindex
- search

## **PYTHON MODULE INDEX**

## а

```
adb_shell, 54
adb_shell.adb_device, 16
adb_shell.adb_device_async, 30
adb_shell.adb_message,44
adb_shell.auth,6
adb_shell.auth.keygen, 1
adb_shell.auth.sign_cryptography, 3
adb_shell.auth.sign_pycryptodome, 3
adb_shell.auth.sign_pythonrsa,4
adb_shell.constants,46
adb_shell.exceptions, 48
adb_shell.hidden_helpers,49
adb_shell.transport, 16
adb_shell.transport.base_transport,6
adb_shell.transport.base_transport_async,
adb_shell.transport.tcp_transport,8
adb_shell.transport.tcp_transport_async,
adb_shell.transport.usb_transport, 10
```

62 Python Module Index

## **INDEX**

Symbols	_banner (adb_shell.adb_device.AdbDeviceUsb at-
_Accum (class in adb_shell.auth.sign_pythonrsa), 5	tribute), 27
_AdbIOManager (class in adb_shell.adb_device), 27	_banner(adb_shell.adb_device_async.AdbDeviceAsync
_AdbIOManagerAsync (class in	attribute), 32
adb_shell.adb_device_async), 41	_banner(adb_shell.adb_device_async.AdbDeviceTcpAsync
_AdbPacketStore (class in	attribute), 41
adb_shell.hidden_helpers), 50	_buf (adb_shell.auth.sign_pythonrsaAccum attribute),
_AdbTransactionInfo (class in	5
adb_shell.hidden_helpers), 52	_clse() (adb_shell.adb_device.AdbDevice method), 18
_FileSyncTransactionInfo (class in	_clse() (adb_shell.adb_device_async.AdbDeviceAsync
adb_shell.hidden_helpers), 53	method), 32
_HANDLE_CACHE (adb_shell.transport.usb_transport.Usb attribute), 12	Transport tion (adb_shell.transport.tcp_transport.TcpTransport attribute), 8
_HANDLE_CACHE_LOCK	_default_transport_timeout_s
(adb_shell.transport.usb_transport.UsbTransport	(adb_shell.adb_device.AdbDevice attribute),
attribute). 12	17
_abc_impl (adb_shell.transport.base_transport.BaseTran	nsportault_transport_timeout_s
attribute), 6	(aab_snett.aab_aevice.AabDevice1cp at-
_abc_impl(adb_shell.transport.base_transport_async.Base_transport_async.	aseTransportAssync <sup>26</sup>
attribute), 7	_deraurt_transport_transport_s
_abc_impl(adb_shell.transport.tcp_transport.TcpTransp attribute), 8	oort (adb_shell.adb_device.AdbDeviceUsb at- tribute), 27
_abc_impl(adb_shell.transport.tcp_transport_async.Tcp	ordefaultsyncansport_timeout_s (adb_shell.adb_device_async.AdbDeviceAsync
attribute), 10	w '1 w \ 22
_abc_impl (adb_shell.transport.usb_transport.UsbTrans	_default_transport_timeout_s
attribute), 12	(adb_shell.adb_device_async.AdbDeviceTcpAsync
_asdict() (adb_shell.hidden_helpers.DeviceFile	attribute), 41
method), 50	_default_transport_timeout_s
_avaliable (ado_sheh.ado_device.habbevice ai	(adb_shell.transport.usb_transport.UsbTransport
tribute), 17	attribute), 11
_available (adb_shell.adb_device.AdbDeviceTcp at-	_device(adb_shell.transport.usb_transport.UsbTransport
tribute), 26 _available (adb_shell.adb_device.AdbDeviceUsb at-	attribute), 11
	_dict (adb_shell.hidden_helpersAdbPacketStore at-
tribute), 27 _available(adb_shell.adb_device_async.AdbDeviceAsy	
attribute), 32	_field_defaults(adb_shell.hidden_helpers.DeviceFile
_available(adb_shell.adb_device_async.AdbDeviceTcp	7
attribute), 41	_fields (adb_shell.hidden_helpers.DeviceFile at-
_banner (adb_shell.adb_device.AdbDevice attribute),	tribute), 50
panner (dab_sheu.dab_device.AdbDevice diiribilie),	_fields_defaults(adb_shell.hidden_helpers.DeviceFile
_banner (adb_shell.adb_device.AdbDeviceTcp at-	attribute), 50
tribute), 26	_filesync_flush()
11 tout ), 20	

```
(adb shell.adb device.AdbDevice method), 18
                                                                                 local id (adb shell.adb device.AdbDeviceUsb at-
_filesync_flush()
                                                                                               tribute), 27
                                                                                 local id(adb shell.adb device async.AdbDeviceAsync
             (adb shell.adb device async.AdbDeviceAsync
             method), 33
                                                                                               attribute), 32
_filesync_read() (adb_shell.adb_device.AdbDevice _local_id(adb_shell.adb_device_async.AdbDeviceTcpAsync
             method), 18
                                                                                               attribute), 41
filesync read() (adb shell.adb device async.AdbDeviceAsyncid lock (adb shell.adb device.AdbDevice
             method), 33
                                                                                               attribute), 17
                                                                                 _local_id_lock (adb_shell.adb_device_async.AdbDeviceAsync
_filesync_read_buffered()
             (adb_shell.adb_device.AdbDevice method), 18
                                                                                               attribute), 32
                                                                                 _make() (adb_shell.hidden_helpers.DeviceFile class
_filesync_read_buffered()
             (adb_shell.adb_device_async.AdbDeviceAsync
                                                                                               method), 50
             method), 33
                                                                                 _max_read_packet_len
_filesync_read_until()
                                                                                               (adb_shell.transport.usb_transport.UsbTransport
             (adb_shell.adb_device.AdbDevice method), 19
                                                                                               attribute), 12
                                                                                 _maxdata (adb_shell.adb_device.AdbDevice attribute),
_filesync_read_until()
             (adb_shell.adb_device_async.AdbDeviceAsync
                                                                                               18
             method), 33
                                                                                 maxdata
                                                                                                            (adb_shell.adb_device.AdbDeviceTcp
_filesync_send()(adb_shell.adb_device.AdbDevice
                                                                                               attribute), 26
             method), 19
                                                                                  maxdata
                                                                                                            (adb shell.adb device.AdbDeviceUsb
_filesync_send() (adb_shell.adb_device_async.AdbDeviceAsynattribute), 27
             method), 34
                                                                                 _maxdata(adb_shell.adb_device_async.AdbDeviceAsync
_find()(adb_shell.transport.usb_transport.UsbTransport
                                                                                               attribute), 32
             class method), 12
                                                                                  maxdata(adb shell.adb device async.AdbDeviceTcpAsync
find and open() (adb shell.transport.usb transport.UsbTranspttribute), 41
             class method), 12
                                                                                  maxdata (adb shell.hidden helpers. FileSyncTransactionInfo
_find_devices()(adb_shell.transport.usb_transport.UsbTransportribute), 53
                                                                                  _okay() (adb_shell.adb_device.AdbDevice method), 19
             class method), 12
_find_first()(adb_shell.transport.usb_transport.Usb<u>Tr</u>ansport)(adb_shell.adb_device_async.AdbDeviceAsync
             class method), 13
                                                                                               method), 34
_flush_buffers()(adb_shell.transport.usb_transport.<u>Usb\Bran\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\text{arth}\sp(\t</u>
             method), 13
                                                                                 _open() (adb_shell.adb_device_async.AdbDeviceAsync
_get_transport_timeout_s()
                                                                                               method), 34
             (adb_shell.adb_device.AdbDevice method), 19
                                                                                 _open() (adb_shell.transport.usb_transport.UsbTransport
_get_transport_timeout_s()
                                                                                               method), 13
             (adb_shell.adb_device_async.AdbDeviceAsync
                                                                                 _packet_store(adb_shell.adb_device._AdbIOManager
             method), 34
                                                                                               attribute), 27
_host (adb_shell.transport.tcp_transport.TcpTransport
                                                                                 _packet_store(adb_shell.adb_device_async._AdbIOManagerAsync
             attribute), 8
                                                                                               attribute), 41
_host (adb_shell.transport.tcp_transport_async.TcpTranspartAsyn(adb_shell.transport.tcp_transport.TcpTransport
             attribute), 9
                                                                                               attribute), 8
_interface_number
                                                                                  _port (adb_shell.transport.tcp_transport_async.TcpTransportAsync
             (adb shell.transport.usb transport.UsbTransport
                                                                                               attribute), 9
                                                                                 _port_path_matcher()
             attribute), 11
_io_manager (adb_shell.adb_device.AdbDevice at-
                                                                                               (adb\_shell.transport.usb\_transport.UsbTransport
             tribute), 17
                                                                                               class method), 13
_io_manager(adb_shell.adb_device_async.AdbDeviceAsypc11()(adb_shell.adb_device.AdbDevice method), 20
             attribute), 32
                                                                                 _pull()(adb_shell.adb_device_async.AdbDeviceAsync
_load_rsa_private_key()
                                                       (in
                                                                    module
                                                                                               method), 35
                                                                                 _push() (adb_shell.adb_device.AdbDevice method), 20
             adb_shell.auth.sign_pythonrsa), 6
                      (adb_shell.adb_device.AdbDevice
                                                                                 _push() (adb_shell.adb_device_async.AdbDeviceAsync
_local_id
             tribute), 17
                                                                                               method), 35
local id (adb shell.adb device.AdbDeviceTcp at-
                                                                                 _read_bytes_from_device()
                                                                                               (adb shell.adb device. AdbIOManager
             tribute), 26
```

```
method), 28
                                                           (adb_shell.adb_device.AdbDevice method), 22
_read_bytes_from_device()
                                                   _streaming_service()
        (adb shell.adb device async. AdbIOManagerAsync
                                                           (adb shell.adb device async.AdbDeviceAsync
        method), 42
                                                           method), 37
_read_endpoint (adb_shell.transport.usb_transport.Usb_transport_ms () (adb_shell.transport.usb_transport.UsbTransport
        attribute), 12
                                                           method), 14
read expected packet from device()
                                                  to bytes () (in module adb shell.auth.keygen), 2
        (adb_shell.adb_device._AdbIOManager
                                                  _transport (adb_shell.adb_device.AdbDeviceTcp at-
        method), 28
                                                           tribute), 26
                                                  _transport (adb_shell.adb_device.AdbDeviceUsb at-
_read_expected_packet_from_device()
        (adb_shell.adb_device_async._AdbIOManagerAsync
                                                           tribute), 27
        method), 42
                                                  _transport (adb_shell.adb_device._AdbIOManager
_read_packet_from_device()
                                                           attribute), 27
        (adb_shell.adb_device._AdbIOManager
                                                  _transport(adb_shell.adb_device_async.AdbDeviceAsync
        method), 28
                                                           attribute), 32
_read_packet_from_device()
                                                  _transport(adb_shell.adb_device_async.AdbDeviceTcpAsync
        (adb_shell.adb_device_async._AdbIOManagerAsync
                                                           attribute), 41
        method), 42
                                                  _transport(adb_shell.adb_device_async._AdbIOManagerAsync
                   (adb shell.adb device.AdbDevice
_read_until()
                                                           attribute), 41
                                                  _transport(adb_shell.transport.usb_transport.UsbTransport
        method), 20
_read_until()(adb_shell.adb_device_async.AdbDeviceAsync
                                                          attribute), 11
        method), 35
                                                  _transport_lock(adb_shell.adb_device._AdbIOManager
_read_until_close()
                                                           attribute), 28
                                                  _transport_lock(adb_shell.adb_device_async._AdbIOManagerAsync
        (adb shell.adb device.AdbDevice method), 21
_read_until_close()
                                                           attribute), 42
                                                  _usb_info(adb_shell.transport.usb_transport.UsbTransport
        (adb_shell.adb_device_async.AdbDeviceAsync
        method), 36
                                                           attribute), 12
_reader(adb_shell.transport.tcp_transport_async.TcpTransportAsyneadpoint(adb_shell.transport.usb_transport.UsbTransport
        attribute), 9
                                                           attribute), 12
                (adb_shell.hidden_helpers.DeviceFile _writer (adb_shell.transport.tcp_transport_async.TcpTransportAsync
replace()
        method), 50
                                                           attribute), 10
              (adb_shell.adb_device._AdbIOManager
_send()
                                                  Α
        method), 29
_send() (adb_shell.adb_device_async._AdbIOManagerAsyms shell (module), 54
        method), 43
                                                  adb_shell.adb_device (module), 16
_serial_matcher()
                                                  adb_shell.adb_device_async (module), 30
        (adb\_shell.transport.usb\_transport.UsbTransport adb\_shell.adb\_message (module), 44
        class method), 13
                                                  adb_shell.auth (module), 6
                   (adb shell.adb device.AdbDevice
service()
                                                  adb_shell.auth.keygen (module), 1
        method), 21
                                                  adb_shell.auth.sign_cryptography
                                                                                              (mod-
_service()(adb_shell.adb_device_async.AdbDeviceAsync
                                                           ule), 3
        method), 36
                                                  adb_shell.auth.sign_pycryptodome
                                                                                              (mod-
_setting(adb_shell.transport.usb_transport.UsbTransport
                                                           ule), 3
        attribute), 12
                                                  adb_shell.auth.sign_pythonrsa(module), 4
_store_lock (adb_shell.adb_device._AdbIOManager
                                                  adb shell.constants (module), 46
        attribute), 27
                                                  adb_shell.exceptions (module), 48
_store_lock(adb_shell.adb_device_async._AdbIOManagersAsymell.hidden_helpers(module), 49
        attribute), 41
                                                  adb_shell.transport (module), 16
_streaming_command()
                                                  adb_shell.transport.base_transport(mod-
        (adb_shell.adb_device.AdbDevice method), 21
                                                           ule), 6
_streaming_command()
                                                  adb_shell.transport.base_transport_async
        (adb_shell.adb_device_async.AdbDeviceAsync
                                                           (module), 7
        method), 36
                                                  adb_shell.transport.tcp_transport (mod-
_streaming_service()
                                                           ule), 8
```

<pre>adb_shell.transport.tcp_transport_async</pre>	<pre>bulk_write() (adb_shell.transport.base_transport_async.BaseTransport method), 7</pre>
adb_shell.transport.usb_transport (mod-ule), 10	bulk_write()(adb_shell.transport.tcp_transport.TcpTransport method), 8
AdbCommandFailureException, 48	bulk_write()(adb_shell.transport.tcp_transport_async.TcpTransport
AdbConnectionError, 48	method), 10
AdbDevice (class in adb_shell.adb_device), 17	<pre>bulk_write() (adb_shell.transport.usb_transport.UsbTransport</pre>
AdbDeviceAsync (class in adb_shell.adb_device_async), 31	method), 14
AdbDeviceTcp (class in adb_shell.adb_device), 26	C
AdbDeviceTcpAsync (class in	can_add_to_send_buffer()
adb_shell.adb_device_async), 40	(adb_shell.hidden_helpersFileSyncTransactionInfo
AdbDeviceUsb (class in adb_shell.adb_device), 26	method), 54
AdbMessage (class in adb_shell.adb_message), 45	checksum() (adb_shell.adb_message.AdbMessage
AdbTimeoutError, 48	property), 45
ANDROID_PUBKEY_MODULUS_SIZE (in module	checksum() (in module adb_shell.adb_message), 46
adb_shell.auth.keygen), 2	CLASS (in module adb_shell.constants), 46
ANDROID_PUBKEY_MODULUS_SIZE_WORDS (in	clear() (adb_shell.hidden_helpersAdbPacketStore
module adb_shell.auth.keygen), 2	method), 50
ANDROID_RSAPUBLICKEY_STRUCT (in module adb_shell.auth.keygen), 2	clear_all() (adb_shell.hidden_helpersAdbPacketStore
arg0 (adb_shell.adb_message.AdbMessage attribute),	method), 51
45	close() (adb_shell.adb_deviceAdbIOManager method), 29
arg1 (adb_shell.adb_message.AdbMessage attribute), 45	close() (adb_shell.adb_device.AdbDevice method), 22 close() (adb_shell.adb_device_asyncAdbIOManagerAsync
<pre>args_match() (adb_shell.hidden_helpersAdbTransac method), 53</pre>	
AUTH_RSAPUBLICKEY (in module	method), 37
adb_shell.constants), 46	<pre>close() (adb_shell.transport.base_transport.BaseTransport</pre>
AUTH_SIGNATURE (in module adb_shell.constants), 46	method), 7
AUTH_TOKEN (in module adb_shell.constants), 46 available() (adb_shell.adb_device.AdbDevice prop-	<pre>close() (adb_shell.transport.base_transport_async.BaseTransportAsyn</pre>
erty), 22	close() (adb_shell.transport.tcp_transport.TcpTransport
available()(adb_shell.adb_device_async.AdbDeviceA	The troub,
property), 37	close() (adb_shell.transport.tcp_transport_async.TcpTransportAsync method), 10
В	<pre>close() (adb_shell.transport.usb_transport.UsbTransport</pre>
BaseTransport (class in adb_shell.transport.base_transport), 6	method), 14 command (adb_shell.adb_message.AdbMessage at-
BaseTransportAsync (class in	tribute), 45
adb_shell.transport.base_transport_async), 7	connect() (adb_shell.adb_deviceAdbIOManager
<pre>bulk_read() (adb_shell.transport.base_transport.Base</pre>	connect() (adb_shell.adb_device.AdbDevice method),
bulk_read() (adb_shell.transport.base_transport_asyn method), 7	c.BaseTransportAsync connect() (adb_shell.adb_device_asyncAdbIOManagerAsync
bulk_read()(adb_shell.transport.tcp_transport.TcpTra	· · · · · · · · · · · · · · · · · · ·
method), 8	connect () (adb_shell.adb_device_async.AdbDeviceAsync
bulk_read() (adb_shell.transport.tcp_transport_async	· · · · · · · · · · · · · · · · · · ·
method), 10	connect () (adb_shell.transport.base_transport.BaseTransport
bulk_read()(adb_shell.transport.usb_transport.UsbTr	
method), 14	connect () (adb_shell.transport.base_transport_async.BaseTransportA
bulk_write()(adb_shell.transport.base_transport.Base	
method), 7	connect () (adb_shell.transport.tcp_transport.TcpTransport method), 9
	тетои), э

```
connect() (adb_shell.transport.tcp_transport_async.TcpFransportAsync(adb_shell.transport.usb_transport.UsbTransport
               method), 10
                                                                                                       class method), 14
connect() (adb shell.transport.usb transport.UsbTransportnd all adb devices()
                                                                                                       (adb\_shell.transport.usb\_transport.UsbTransport
              method), 14
CryptographySigner
                                                        (class
                                                                                  in
                                                                                                       class method), 15
              adb shell.auth.sign cryptography), 3
                                                                                        find allow zeros()
                                                                                                       (adb shell.hidden helpers. AdbPacketStore
D
                                                                                                       method), 51
                                                                                        FromRSAKeyPath()(adb_shell.auth.sign_pythonrsa.PythonRSASigner
data (adb_shell.adb_message.AdbMessage attribute),
                                                                                                       class method), 5
                                                   (in
decode pubkey()
                                                                          module
                                                                                        G
              adb_shell.auth.keygen), 2
decode pubkey file()
                                                         (in
                                                                          module
                                                                                        get()
                                                                                                            (adb_shell.hidden_helpers._AdbPacketStore
              adb_shell.auth.keygen), 2
                                                                                                       method), 51
DEFAULT_AUTH_TIMEOUT_S
                                                                                        get_banner() (in module adb_shell.hidden_helpers),
                                                           (in
                                                                          module
              adb_shell.constants), 46
                                                                                                       54
                                                                                        get_files_to_push()
DEFAULT_PUSH_MODE
                                                     (in
                                                                                                                                                                  module
                                                                          module
                                                                                                       adb_shell.hidden_helpers), 54
              adb_shell.constants), 47
DEFAULT_READ_TIMEOUT_S
                                                           (in
                                                                          module
                                                                                        get interface()
                                                                                                                                                                  module
                                                                                                       adb_shell.transport.usb_transport), 15
              adb_shell.constants), 47
DEFAULT_TIMEOUT_S
                                                                                        get_user_info()
                                                                                                                                                                  module
                                                     (in
                                                                          module
                                                                                                                                            (in
              adb_shell.transport.usb_transport), 11
                                                                                                       adb shell.auth.keygen), 2
DeviceAuthError, 48
                                                                                        GetPublicKey()(adb_shell.auth.sign_cryptography.CryptographySign
DeviceFile (class in adb_shell.hidden_helpers), 50
                                                                                                       method), 3
DevicePathInvalidError, 48
                                                                                        GetPublicKey()(adb_shell.auth.sign_pycryptodome.PycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.sign_pycryptodomeAuth.s
                         (adb_shell.auth.sign_pythonrsa._Accum
                                                                                                       method), 4
digest()
              method), 5
                                                                                        GetPublicKey() (adb_shell.auth.sign_pythonrsa.PythonRSASigner
                                                                                                       method), 5
E
                                                                          module
encode_pubkey()
                                                   (in
                                                                                        ID_TO_WIRE (in module adb_shell.constants), 47
              adb_shell.auth.keygen), 2
                                                                                        IDS (in module adb_shell.constants), 47
exec_out()
                                  (adb_shell.adb_device.AdbDevice
                                                                                        int_to_cmd() (in module adb_shell.adb_message),
              method), 23
exec_out() (adb_shell.adb_device_async.AdbDeviceAsync
              method), 38
                                                                                        interface_matcher()
                                                                                                                                                (in
                                                                                                                                                                  module
                                                                                                       adb_shell.transport.usb_transport), 15
F
                                                                                        InvalidChecksumError, 48
                                                                                        InvalidCommandError, 48
                            (adb shell.hidden helpers.DeviceFile
filename()
                                                                                        InvalidResponseError, 48
              property), 50
                                                                                        InvalidTransportError, 48
FILESYNC_ID_TO_WIRE
                                                       (in
                                                                          module
              adb_shell.constants), 47
FILESYNC_IDS (in module adb_shell.constants), 47
                                                                                        keygen() (in module adb_shell.auth.keygen), 2
FILESYNC_LIST_FORMAT
                                                                          module
                                                        (in
              adb_shell.constants), 47
FILESYNC_PULL_FORMAT
                                                         (in
                                                                          module
                                                                                        list() (adb_shell.adb_device.AdbDevice method), 23
              adb_shell.constants), 47
                                                                                        list() (adb_shell.adb_device_async.AdbDeviceAsync
FILESYNC_PUSH_FORMAT
                                                         (in
                                                                          module
              adb_shell.constants), 47
                                                                                                       method), 38
                                                                                        local id (adb shell.hidden helpers. AdbTransactionInfo
FILESYNC STAT FORMAT
                                                        (in
                                                                          module
                                                                                                       attribute), 52
              adb_shell.constants), 47
FILESYNC_WIRE_TO_ID
                                                       (in
                                                                          module
                                                                                        M
               adb_shell.constants), 47
                   (adb_shell.hidden_helpers._AdbPacketStore
                                                                                        magic (adb_shell.adb_message.AdbMessage attribute),
find()
              method), 51
                                                                                                       45
```

```
MAX ADB DATA (in module adb shell.constants), 47
                                                      recv message size
MAX CHUNK SIZE (in module adb shell.constants), 47
                                                               (adb shell.hidden helpers. FileSyncTransactionInfo
max chunk size() (adb shell.adb device.AdbDevice
                                                               attribute), 53
                                                      remote_id(adb_shell.hidden_helpers._AdbTransactionInfo
        property), 24
max chunk size() (adb shell.adb device async.AdbDeviceAsynattribute), 52
                                                      root () (adb shell.adb device.AdbDevice method), 24
        property), 38
MAX PUSH DATA (in module adb shell.constants), 47
                                                      root() (adb shell.adb device async.AdbDeviceAsync
MESSAGE FORMAT (in module adb shell.constants), 47
                                                               method), 39
                                                      \verb|rsa_key| (adb\_shell.auth.sign\_cryptography.CryptographySigner|
MESSAGE SIZE (in module adb shell.constants), 47
mode() (adb_shell.hidden_helpers.DeviceFile prop-
                                                               attribute), 3
         erty), 50
                                                      rsa_key(adb_shell.auth.sign_pycryptodome.PycryptodomeAuthSigner
mtime() (adb_shell.hidden_helpers.DeviceFile prop-
                                                               attribute), 4
        erty), 50
                                                      S
Р
                                                      send()
                                                                     (adb_shell.adb_device._AdbIOManager
pack() (adb shell.adb message.AdbMessage method),
                                                               method), 30
                                                      send()(adb_shell.adb_device_async._AdbIOManagerAsync
port_path() (adb_shell.transport.usb_transport.UsbTransport
                                                              method), 44
        property), 15
                                                      send_buffer(adb_shell.hidden_helpers._FileSyncTransactionInfo
priv_key(adb_shell.auth.sign_pythonrsa.PythonRSASigner
                                                               attribute), 53
        attribute), 5
                                                      send idx (adb shell.hidden helpers. FileSyncTransactionInfo
PROTOCOL (in module adb shell.constants), 47
                                                               attribute), 54
pub_key (adb_shell.auth.sign_pythonrsa.PythonRSASigneserial_number() (adb_shell.transport.usb_transport.UsbTransport
         attribute), 5
                                                              property), 15
public_key (adb_shell.auth.sign_cryptography.Cryptographe/Signer(adb_shell.adb_device.AdbDevice method), 25
         attribute), 3
                                                      shell() (adb_shell.adb_device_async.AdbDeviceAsync
public_key (adb_shell.auth.sign_pycryptodome.PycryptodomeAuthSathwef), 39
         attribute), 4
                                                      Sign () (adb_shell.auth.sign_cryptography.CryptographySigner
pull() (adb_shell.adb_device.AdbDevice method), 24
                                                               method), 3
pull() (adb_shell.adb_device_async.AdbDeviceAsync
                                                      Sign()(adb\_shell.auth.sign\_pycryptodome.PycryptodomeAuthSigner)
         method), 38
                                                               method), 4
push() (adb_shell.adb_device.AdbDevice method), 24
                                                      Sign() (adb_shell.auth.sign_pythonrsa.PythonRSASigner
push() (adb_shell.adb_device_async.AdbDeviceAsync
                                                              method), 5
        method), 39
                                                                (adb_shell.hidden_helpers.DeviceFile prop-
                                                      size()
PushFailedError, 48
                                                               erty), 50
            (adb_shell.hidden_helpers._AdbPacketStore
                                                      stat() (adb_shell.adb_device.AdbDevice method), 25
put()
        method), 51
                                                      stat() (adb_shell.adb_device_async.AdbDeviceAsync
PycryptodomeAuthSigner
                                                 in
                                                               method), 40
                                     (class
        adb shell.auth.sign pycryptodome), 4
                                                      streaming shell()
PythonRSASigner
                                                               (adb_shell.adb_device.AdbDevice method), 25
                                (class
                                                 in
        adb_shell.auth.sign_pythonrsa), 4
                                                      streaming shell()
                                                               (adb_shell.adb_device_async.AdbDeviceAsync
R
                                                               method), 40
                                                      SUBCLASS (in module adb_shell.constants), 48
read()
               (adb_shell.adb_device._AdbIOManager
         method), 30
read() (adb_shell.adb_device_async._AdbIOManagerAsync
         method), 44
                                                      TcpTimeoutException, 49
read timeout s (adb shell.hidden helpers. AdbTransactionIbnfonsport
                                                                                    (class
                                                                                                       in
        attribute), 52
                                                               adb shell.transport.tcp transport), 8
recv_buffer(adb_shell.hidden_helpers._FileSyncTransactionhafonsportAsync
                                                                                        (class
                                                                                                       in
         attribute), 53
                                                               adb_shell.transport.tcp_transport_async),
recv_message_format
         (adb_shell.hidden_helpers._FileSyncTransactionInfomeout_s (adb_shell.hidden_helpers._AdbTransactionInfo
         attribute), 53
                                                               attribute), 53
```

```
transport_timeout_s
        (adb_shell.hidden_helpers._AdbTransactionInfo
        attribute), 53
U
unpack() (in module adb_shell.adb_message), 46
              (adb_shell.auth.sign_pythonrsa._Accum
update()
        method), 6
usb_error (adb_shell.exceptions.UsbReadFailedError
        attribute), 49
usb_info() (adb_shell.transport.usb_transport.UsbTransport
        property), 15
UsbDeviceNotFoundError, 49
UsbReadFailedError, 49
UsbTransport
                                               in
        adb_shell.transport.usb_transport), 11
UsbWriteFailedError, 49
VERSION (in module adb_shell.constants), 48
W
WIRE_TO_ID (in module adb_shell.constants), 48
write_public_keyfile()
                                  (in
                                           module
        adb_shell.auth.keygen), 3
```