

file:message-capture-parser

Link

[bitcoin/contrib/message-capture/message-capture-parser.py_](https://github.com/bitcoin/bitcoin/blob/master/contrib/message-capture/message-capture-parser.py#L1)
[at e25af11225d9d94ecf7068bf7a9a359268786fbe ·](https://github.com/bitcoin/bitcoin/blob/master/contrib/message-capture/message-capture-parser.py#L1)
[bitcoin/bitcoin · GitHub](https://github.com/bitcoin/bitcoin)

▼ Code

```
import argparse
import os
import shutil
import sys
from io import BytesIO
import json
from pathlib import Path
from typing import Any, List, Optional

sys.path.append(os.path.join(os.path.dirname(__file__), '../.

from test_framework.messages import ser_uint256      # noqa: E
from test_framework.p2p import MESSAGE_MAP           # noqa: E

TIME_SIZE = 8
LENGTH_SIZE = 4
MSGTYPE_SIZE = 12

HASH_INTS = [
    "blockhash",
    "block_hash",
    "hash",
```

```

    "hashMerkleRoot",
    "hashPrevBlock",
    "hashstop",
    "prev_header",
    "sha256",
    "stop_hash",
]

HASH_INT_VECTORS = [
    "hashes",
    "headers",
    "vHave",
    "vHash",
]

class ProgressBar:
    def __init__(self, total: float):
        self.total = total
        self.running = 0

    def set_progress(self, progress: float):
        cols = shutil.get_terminal_size()[0]
        if cols <= 12:
            return
        max_blocks = cols - 9
        num_blocks = int(max_blocks * progress)
        print('\r[ {}{} ] {:.3.0f}%'.format('#' * num_blocks,
            ' ' * (max_blocks - num_blocks),
            progress * 100),
            end='')

    def update(self, more: float):
        self.running += more
        self.set_progress(self.running / self.total)

```

```

def to_jsonable(obj: Any) -> Any:
    if hasattr(obj, "__dict__"):
        return obj.__dict__
    elif hasattr(obj, "__slots__"):
        ret = {}    # type: Any
        for slot in obj.__slots__:
            val = getattr(obj, slot, None)
            if slot in HASH_INTS and isinstance(val, int):
                ret[slot] = ser_uint256(val).hex()
            elif slot in HASH_INT_VECTORS and all(isinstance(
                ret[slot] = [ser_uint256(a).hex() for a in va
            else:
                ret[slot] = to_jsonable(val)
        return ret
    elif isinstance(obj, list):
        return [to_jsonable(a) for a in obj]
    elif isinstance(obj, bytes):
        return obj.hex()
    else:
        return obj

def process_file(path: str, messages: List[Any], recv: bool,
    with open(path, 'rb') as f_in:
        if progress_bar:
            bytes_read = 0

        while True:
            if progress_bar:
                diff = f_in.tell() - bytes_read - 1
                progress_bar.update(diff)
                bytes_read = f_in.tell() - 1

            tmp_header_raw = f_in.read(TIME_SIZE + LENGTH_SIZ

```

```

if not tmp_header_raw:
    break
tmp_header = BytesIO(tmp_header_raw)
time = int.from_bytes(tmp_header.read(TIME_SIZE),
msgtype = tmp_header.read(MSGTYPE_SIZE).split(b'\
length = int.from_bytes(tmp_header.read(LENGTH_SI

msg_dict = {}
msg_dict["direction"] = "recv" if recv else "sent
msg_dict["time"] = time
msg_dict["size"] = length

msg_ser = BytesIO(f_in.read(length))

if msgtype not in MESSAGEMAP:
    try:
        msgtype_tmp = msgtype.decode()
        if not msgtype_tmp.isprintable():
            raise UnicodeDecodeError
        msg_dict["msgtype"] = msgtype_tmp
    except UnicodeDecodeError:
        msg_dict["msgtype"] = "UNREADABLE"
    msg_dict["body"] = msg_ser.read().hex()
    msg_dict["error"] = "Unrecognized message typ
    messages.append(msg_dict)
    print(f"WARNING - Unrecognized message type {
    continue

msg = MESSAGEMAP[msgtype]()
msg_dict["msgtype"] = msgtype.decode()

try:
    msg.deserialize(msg_ser)
except KeyboardInterrupt:
    raise
except Exception:

```

```

        msg_ser.seek(0, os.SEEK_SET)
        msg_dict["body"] = msg_ser.read().hex()
        msg_dict["error"] = "Unable to deserialize me
        messages.append(msg_dict)
        print(f"WARNING - Unable to deserialize messa
        continue

    if length:
        msg_dict["body"] = to_jsonable(msg)
        messages.append(msg_dict)

    if progress_bar:
        f_in.seek(0, os.SEEK_END)
        diff = f_in.tell() - bytes_read - 1
        progress_bar.update(diff)

def main():
    parser = argparse.ArgumentParser(
        description=__doc__,
        epilog="EXAMPLE \n\t{0} -o out.json <data-dir>/messag
        formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument(
        "capturepaths",
        nargs='+',
        help="binary message capture files to parse.")
    parser.add_argument(
        "-o", "--output",
        help="output file. If unset print to stdout")
    parser.add_argument(
        "-n", "--no-progress-bar",
        action='store_true',
        help="disable the progress bar. Automatically set if
    args = parser.parse_args()
    capturepaths = [Path.cwd() / Path(capturepath) for captur
    output = Path.cwd() / Path(args.output) if args.output el

```

```

use_progress_bar = (not args.no_progress_bar) and sys.std

messages = []
if use_progress_bar:
    total_size = sum(capture.stat().st_size for capture in capturepaths)
    progress_bar = ProgressBar(total_size)
else:
    progress_bar = None

for capture in capturepaths:
    process_file(str(capture), messages, "recv" in capture.options)

messages.sort(key=lambda msg: msg['time'])

if use_progress_bar:
    progress_bar.set_progress(1)

jsonrep = json.dumps(messages)
if output:
    with open(str(output), 'w+', encoding="utf8") as f_out:
        f_out.write(jsonrep)
else:
    print(jsonrep)

if __name__ == "__main__":
    main()

```

▼ Documentation

Per-Peer Message Capture

Purpose

This feature allows for message capture on a per-peer basis.

Usage and Functionality

- * Run `bitcoind` with the `-capturemessages` option.
- * Look in the `message_capture` folder in your datadir.
 - * Typically this will be `~/.bitcoin/message_capture`.
 - * See that there are many folders inside, one for each peer
 - * Inside each peer's folder there are two `.dat` files: one
- * Run `contrib/message-capture/message-capture-parser.py` with
 - * See the `-h` option for help.
 - * To see all messages, both sent and received, for all peers
...
./contrib/message-capture/message-capture-parser.py -o out
~/.bitcoin/message_capture/**/* .dat
...
* Note: The messages in the given `.dat` files will be interpreted as JSON.
* If an output file is not provided (i.e. the `-o` option is not used), the output will be printed to stdout.
* View the resulting output.
 - * The output file is `JSON` formatted.
 - * Suggestion: use `jq` to view the output, with `jq . out.json`

Prompt

As a code documentation assistant, you are responsible for documenting at the file/script level. When provided a file level code, your approach involves adding a header comment at the top of the file. This comment should be the documentation for the code and include all relevant information needed to understand or use the script. The code is provided below.

Responses

ChatGPT

```
"""
```

```
File-Level Code Documentation
```

```
This script is responsible for processing binary message capture  
It provides functionality to parse these files, deserialize mes
```

```
Author: [Author's Name]
```

```
Email: [Author's Email]
```

```
Date: [Date of Creation/Last Update]
```

```
Usage:
```

```
python script_name.py <capturepaths> [-o output_file] [-n]
```

```
Arguments:
```

```
<capturepaths>: List of binary message capture files to pars
```

```
Options:
```

```
-o, --output: Specify the output file for the JSON represent
```

```
-n, --no-progress-bar: Disable the progress bar. It is autor
```

```
Dependencies:
```

- argparse
- os
- shutil
- sys
- io.BytesIO
- json
- pathlib.Path
- typing.Any
- typing.List
- typing.Optional

- test_framework.messages.ser_uint256
- test_framework.p2p.MESSAGEMAP

Constants:

- TIME_SIZE: Size of the time field in bytes (8 bytes).
- LENGTH_SIZE: Size of the length field in bytes (4 bytes).
- MSGTYPE_SIZE: Size of the message type field in bytes (12 bytes).
- HASH_INTS: List of string keys representing hash-related constants.
- HASH_INT_VECTORS: List of string keys representing hash-related vectors.

Classes:

- ProgressBar: A class to display a progress bar for processing files.
 - __init__(self, total: float): Constructor with the total progress.
 - set_progress(self, progress: float): Set the progress.
 - update(self, more: float): Update the progress with an additional amount.

Functions:

- to_jsonable(obj: Any) -> Any: Convert an object to a JSON-serializable format.
- process_file(path: str, messages: List[Any], recv: bool, progress_bar: Optional[ProgressBar]) -> List[Any]:
 - path: Path to the capture file.
 - messages: List to store message dictionaries.
 - recv: True if the messages are received, False if sent.
 - progress_bar: Optional progress bar for displaying progress.
- main(): The main function to parse capture files and generate a report.

This script is designed to parse binary message capture files, (e.g., .bin files).

"""

GPT4

```

"""
Script Title: Message Capture File Parser

Description:
This script is designed to parse binary message capture files, (

Author: [Your Name]
Creation Date: [Creation Date]
Modification Date: [Last Modification Date]

Usage:
Run this script with the command `python [script_name].py -o out

Dependencies:
- Python Standard Library Modules: argparse, os, shutil, sys, io
- Custom Modules: test_framework.messages, test_framework.p2p

Input/Output:
- Input: Binary message capture files (*.dat)
- Output: JSON file containing deserialized message information

Error Handling:
Handles unrecognized or undecodable message types with warnings

License: [Specify License Here]

Additional Notes:
- The progress bar functionality automatically disables if the (
- The script sorts messages by their timestamp before outputting
"""

```

Bard

This code parses bitcoin message capture files.

It takes a list of capture files as input and outputs a JSON file.

direction: "recv" for messages received from the peer, "sent" for messages sent to the peer

time: the timestamp of the message in seconds since the Unix epoch

size: the size of the message in bytes

msgtype: the type of the message, e.g. "version", "getblocks", "getdata"

body: the body of the message, in JSON format

error: an error message if the message could not be parsed

The output JSON file can be used to analyze the traffic between peers.

Here is an example of how to use the script:

```
./message_capture_parser.py -o out.json data-dir/message_capture_files
```

This will parse all of the message capture files in the data-dir and output the results to out.json.

LLama2

This script parses binary message capture files generated by `tcpdump`.

Here's a breakdown of the code:

1. The first section imports various libraries, including `argparse`.
2. The next section defines a custom exception class called `MessageCaptureError`.
3. The `process_file()` function takes a path to a binary message capture file and processes it.
4. The `parse_message()` function takes a byte string representing a message and parses it.
5. The `main()` function is the entry point of the program. It parses the command line arguments and calls the `process_file()` function.
6. The last line of the script runs the `main()` function if the script is being run directly.

StarChat

```
"""
```

```
Script Name: my_awesome_script.py
```

Description: A short summary describing what the script does.

Usage: python my_awesome_script.py --help

Arguments:

arg1 (required): Description of argument 1.

arg2 (optional): Description of argument 2. Defaults to somet

Dependencies:

- package1 >= version1

- package2 ~= version2

Notes:

- Make sure to do thing first before doing other things!

- Remember to handle exceptions gracefully.

Example:

```
>>> python my_awesome_script.py arg1=value1 arg2=value2
```

License: MIT License

Author: Your name <<EMAIL>> (<https://your-website.com/>)

"""