aiowintest Documentation

Release 0.0.2.

Hans SM0UTY

Nov 14, 2018

Contents

Version: 0.0.2.

CHAPTER 1

Client API

The client API uses asyncio and is designed to be easy to use. The API is far from finished and expect it to be changed without notice! You have been warned!

Typical use cases are mainly bridging information from and to Win-Test. At http://sk0ux.se/ we intend to use it to display current scores on a monitor in the club house, and also bridging gab messages to/from our instant messaging system.

Currently the functionality is limited to sending/receiving gab messages and score board summary.

1.1 Installation

As aiowintest uses asyncio, it requres python 3. Python 3.6 and 3.7 have been tested but older versions should work, or be easy to get to work. Pull requests are appreciated!

Install aiowintest using pip:

```
$ pip install aiowintest
```

1.2 Quickstart

A quick example of how to receive gab-messages from Win-Test:

```
import asyncio
import aiowintest

# replace the address below with your network broadcast address
broadcast_addr = ('192.168.11.255', 9000)
local_addr = ('0.0.0.0', 9871)

async def on_gab(message):
```

(continues on next page)

(continued from previous page)

```
    print(message)

async def main(argv):
    loop = asyncio.get_event_loop()
    wt = WintestProtocol(loop, local_addr, broadcast_addr)
    wt.add_handler('gab', on_gab)

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(sys.argv))
    loop.run_forever()
```

Please note that the machine running the code above *must* be on the same network as the Win-Test machine(s).

CHAPTER 2

The Win-Test Protocol

The Win-Teat protocol is based on UDP broadcast packets, by default sent to the local broadcast address at port 9871.

There is some partial documentation of the Win-Test protocol here: http://download.win-test.com/utils/SummaryBroadcastingSpecs.txt

This protocol documentation was made by reading the documentation above, and by reverse engineering the protocol using Wireshark and python code.

The resulting documentation, python code or anything else has no affiliation with the Win-Test developers. It has been created out of personal curiosity and is presented in the hope that it may be useful or at least amusing to others.

Any further insights and/or corrections to this project is greatly appreciated. Please contact me here: https://github.com/hin/aiowintest

2.1 Payload structure

The payload structure is mostly an ASCII-encoded string followed by a checksum and an optional null-character.

The structure of the payload is as follows:

field	type
frame_type	string
:	colon delimiter (ascii 58)
space	space delimiter (ascii 32)
from_station	string
space	space delimiter (ascii 32)
to_station	string
space	space delimiter (ascii 32)
data[0]	data field
space	space delimiter (ascii 32)
data[1]	data field
space	space delimiter (ascii 32)
...	
data[n]	data field
checksum	byte
null	null character with unknown purpose (optional)

For some packets, the payload also contains a null (0x00) character at the end. According to the limited protocol documentation I've found, it should not be there, and I've not been able to find any purpose of it. It may simply be a bug in the Win-Test software.

For now, I check for the extra null character and ignore it if it exists.

2.2 Checksum calculation

When sending a packet, the checksum is calculated as: (sum of all bytes) | 128 % 256

In python, this can be calculated as:

```
def checksum(data):
    return (sum(data)|128)%256
```

2.3 Data field format and encoding

The data fields in the payload are either strings enclosed in quotes (ascii 34) or integers encoded as decimal strings without quotes.

If a string contains the quote character, it is escaped with a backslash (ascii 92).

Win-Test seems to accept iso8859-1 and any character code above 127 is encoded as a backslash followed by the character code in octal.

For example, the string åäö" is encoded as:

original character	encoded characters	ascii code
å	backslash,3,4,5	92,51,52,53
ä	backslash,3,4,4	92,51,52,52
ö	backslash,3,6,6	92,51,54,54
"	backslash,"	92,34

2.4 Frame types

The frame_type field in the packets specifies what kind of information the packet contains. I have not found a list of *all* the frame types so this document contains what can be found in the official docs, as well as the packets I've encountered while reverse engineering the protocol.

2.4.1 Frame Type GAB

Gab messages are the chat messages that can be sent between stations. Apart from the from_station and to_station, it only contains one data field, which contains the actual text of the gab message.

The to_station is a zero length string in the case of a gab message that is sent to all stations.

2.4.2 Frame Type SUMMARY

This packet contains information about the score status of the contest. It is described in detail in the official document linked to above.

2.4.3 Frame Type RCVDPKT

RCVDPKT is used to broadcast DX cluster spots coming from the telnet connection. The from_station is 'TELNET' and the to_station is an zero length string. There is one data field which is formatted as it arrives over telnet from the DX cluster.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search