## C64 Debugger pipe integration specs

version 2019/11/28

Injecting asynchronous commands to an already running instance of Retro Debugger is also possible via standard pipe mechanism. When application is started and initialisation is finished two files for communication will be created:

For sending input (requests) to Retro Debugger:

/tmp/c64debugger-<pid>-in.pipe

For getting output (results) from Retro Debugger:

/tmp/c64debugger-<pid>-out.pipe

Where <pid> is a process id of the running Retro Debugger, for example:

/tmp/c64debugger-1234-in.pipe

/tmp/c64debugger-1234-out.pipe

Pipes are processed asynchronously in a standard FIFO manner and are in binary format.

Packets have a header of 4 bytes magic followed by 4 bytes 32-bit value meaning total size of packet afterwards, then 2 bytes packet identifier, followed by 4 bytes 32-bit value meaning sequence number and then packet data.

Starting marker of packet (a magic) is 4 bytes array:

C0 DE FE ED

Example packet:

C0 DE FE ED 00 00 00 04 00 01 00 00 00 01 AA BB

packet magic packet size packet id sequence number packet data

Packet size is total size of packet without 8 bytes for packet magic and packet size values, meaning it is packet data size + 6 bytes (2 bytes for packet id and 4 bytes for sequence number).

Output packets have the same format. Note, that the sequence number is used to synchronize request packets and their results, i.e. result packet of a packet that requested data will contain the same sequence number. If the packet is out of sequence it will have a sequence number set to FF FF FF FF.

Request Packets specification (in):

| Packet ID | Packet data |
| --- | --- |
| 0000 | Data confirmation status: 1 byte: status code  00 = OK  01 = Data not valid (f.e. file not found,   address out of bounds, etc) |
| 0001 | Load file  1 byte: should auto start? 00=false 01=true 1 byte: should show loading address 1 byte: force reset after load 4 bytes: size of path string string: path to file in UTF16 (2 bytes per char) |
| 0002 | Pause running |
| 0003 | Continue running |
| 0004 | Step one cycle |
| 0005 | Step one instruction |
| 0006 | Step over JSR instruction |
| 0007 | Soft reset C64 |
| 0008 | Hard reset C64 |
| 0009 | JMP to address at cursor |
| 000A | Move cursor to address 2 bytes: address |
| 000B | Make jump to address (set PC) 2 bytes: address |
| 000C | Get CPU status. Result will be sent in the data out pipe:  Result packet:  2 bytes: PC address  1 byte: value of A register  1 byte: value of X register  1 byte: value of Y register  1 byte: value of SP register  1 byte: value of $0001  1 byte: value of P register  1 byte: CPU cycle  1 byte: VIC cycle  2 bytes: raster Y position  2 bytes: raster X position  1 byte: value of EG |
| 000D | Set break on PC at address2 bytes: address 1 byte: set breakpoint 00=off 01=on |