C64GameStation SDK

C64GameStation is a system that allows Commodore64 games to be created/converted to use online storage in place of a disk drive. It operates much like a real disk drive and can accommodate multi-disk games and even games with sector based loading.

C64GameStation is supplied as an Easyflash cartridge that contains all the networking functionality used by the API in this document.

Network connectivity is done via a WiC64 cartridge, so both a WiC64 and Easyflash are required for use on real C= hardware. For emulation, Vice and Kernal64 both have the necessary support.

Games for GameStation are published as a zip file. There are two ways to create valid zip files, Image Mode and File Mode. Regardless of the mode, Zip file must be the name of the game, in 10 characters or less. In addition to whatever game files are in the zip, there must be one .txt file. The name of the text file will be the name used in the C64GameStation menu. This gives you (max) 35 characters for the presentation of the game name. For example: “Double Dragon.txt” inside ddragon1.zip will display “Double Dragon” in the GameStation Menu.

IMAGE MODE:

Image mode operates on standard D64 files and is uploaded as a zip file containing up to 10 .d64 files. Using this method, you call the InsertDisk method as necessary, and load files directly from the appropriate d64 image inside the zip file. This method allows minimal changes to a game for fastest implementation.

D64 files inside the zip must simply be named the disk#. (ie: 1.d64, 2.d64, etc). GameStation will load and execute the first file on disk #1 in the zip file by performing a jump to $080d. The first file must start properly in this manner. This is well suited to self-extracting Exomized files.

If the zip file contains any .d64 images, image mode will be set sutomatically.

FILE MODE:

In file mode, game files (.prg, etc) are directly stored in the zip file. In this method, disk images are not necessary, you can simply specify the filename and call load. No limits in number of files. This mode is useful for large games and cartridge conversions. In this mode, your zip file must contain a file named “gsstartup”. This will be the file that is loaded and executed by GameStation at startup. Just like in D64 mode, this file should start properly from a jump to $080d.

Track/Sector loading is supported in zip mode by saving each track as a file and including it in the zip. Each track file must be named according to disk and track, 4 digits total, first two digits is disk#, second two digits is track#.

For example, for Disk 2, track 10, the filename would be “0210”. The appropriate sector will be returned from the track file.

COMMAND DESCRIPTIONS

CMD_INSERTDISK

This is called to switch disks. Put disk # (0-9) in reg X before call.

This command adds #$30 (for petscii) and stores result in GS_DiskNumber.

```
ldx #2

lda #CMD_INSERTDISK

jsr Execute_Command
```

CMD_LOADFILE

Loads a file to memory. GS_FILENAME should be set first (with trailing 0 byte if <10 chars).

There are three "modes" used by CMD_LOADFILE depending on the data being loaded. See GS_DataFormat below for more info.

```
lda #"T"

sta GS_FILENAME

lda #"E"

sta GS_FILENAME+1

lda #"S"

sta GS_FILENAME+2

lda #"T"
```

```
sta GS_FILENAME+3
lda #0
sta GS_FILENAME+4
lda #CMD_LOADFILE
jsr Execute_Command
```

CMD_LOADSECTOR

Loads the track/sector specified in X/Y to address set with CMD_SETDESTADDRESS.

```
ldx #$01 ;track
ldy #$10 ;sector
lda #CMD_LOADSECTOR
jsr Execute_Command
```

CMD_STREAMDECRUNCH

Loads/decrunches (levelpacked) exomized file.

All files loaded by GSCMD_StreamDecrunch command must be levelpacked using Exomizer v3.0.2 or compatible. Files are streamed from the web and decrunched into the destination address.

```
lda #"T"
sta GS_FILENAME
lda #"E"
sta GS_FILENAME+1
lda #"S"
sta GS_FILENAME+2
lda #"T"
sta GS_FILENAME+3
lda #0
```

```
sta GS_FILENAME+4
lda #CMD_STREAMDECRUNCH
jsr Execute_Command
```

CMD_STREAMFILE

Opens a file stream for reading byte by byte.

```
lda #"T"
sta GS_FILENAME
lda #"E"
sta GS_FILENAME+1
lda #"S"
sta GS_FILENAME+2
lda #"T"
sta GS_FILENAME+3
lda #0
sta GS_FILENAME+4
lda #CMD_STREAMFILE
jsr Execute_Command
```

CMD_STREAMSECTOR

Opens a sector stream for reading byte by byte.

```
ldx #$01 ;track
ldy #$10 ;sector
lda #CMD_STREAMFILE
jsr Execute_Command
```

CMD_GETSTREAMBYTE

Gets a byte from an open stream. Result byte is returned in X.

```
lda #CMD_GETSTREAMBYTE

jsr Execute_Command

txa ;Transfer to A if desired.
```

CMD_GETPALNTSC

CheckPALNTSC - Result in X, 0=pal,1=ntsc

```
lda #CMD_GETPALNTSC

jsr Execute_Command
```

CMD_SETDESTADDRESS

Sets the destination address for load operations.

```
ldx #<$1000

ldy #>$1000

lda #CMD_SETDESTADDRESS

jsr Execute_Command
```

CMD_RESETWIC

Re-Initialize WiC

```
lda # CMD_RESETWIC

jsr Execute_Command
```

CMD_INIT

This is the cart startup routine, probably not used by your game code.

```
lda # CMD_INIT
jsr Execute_Command
```

Execute_Command

Executes one of the above commands in regA.

```
jsr Execute_Command
```

C64GAMESTATION VARIABLES

VAR_DISK

Current Disk#, set with CMD_INSERTDISK.

VAR_SECTOR

Current Sector#, set with various track/sector commands.

VAR_TRACK

Current Track#, set with various track/sector commands.

GS_GAMENAME

Game name, used by the C64GameStation menu.

GS_FILENAME

Current filename for use with load commands.

GS_TIMEOUT

Network timeout value. You should not change this value :)

GS_DataFormat

GS_DataFormat determines how the load operations handle the first two bytes of a file that is loaded.

Options:

0 = Load first two bytes as data.

This is used for example when loading from sectors.

1 = Use first two bytes as load address.

When set to 1, data is loaded two the address at the first two bytes.

2 = Skip first two bytes.

This skips the first two bytes, data is loaded to address set with CMD_SETDESTADDRESS.

HOW TO USE

Include the "*C64GameStation.inc*" file in your source:

All commands are set in regA before calling Execute_Command.

All the examples in this document do not take interrupts into consideration, which in the real world you will likely need to.

If you are having issues getting the commands to work, you may need to surround calls to Execute_Command with sei/cli...

Kernal Emulation

Common kernal loading methods are emulated by C64GameStation. This makes converting keral loading games much easier.

The available functions are:

- FFD5
- FFC0
- FFCF

To use these, you need to create small function wrappers in your code, then repoint the vectors to your wrappers. Typically the wrapper functions will look like this:

```
ffd5
        lda #CMD_FFD5
        sei
        jsr Execute_Command
        cli
        rts

ffcf
        stx ffcfx
        lda #CMD_GETSTREAMBYTE
        sei
        jsr Execute_Command
        cli
        txa
ffcfx=*+1
        ldx #00
        clc
        rts
```

```
ffc0

            lda #CMD_FFC0

            sei

            jsr Execute_Command

            cli

            clc

            rts
```

With those in-place, repoint the vectors and you are done.

```
;Repoint kernal load vectors

            lda #<ffd5

            sta $0330

            lda #>ffd5

            sta $0331


            lda #<ffcf

            sta $0324

            lda #>ffcf

            sta $0325


            lda #<ffc0

            sta $031a

            lda #>ffc0

            sta $031b
```

Questions? stephenody@gmail.com