SPARKLE v1.1

Sparkle is a trackmo loader solution for the Commodore 64 inspired by Lft's Spindle and Krill's loader. It utilizes full on-the-fly GCR processing, fast data transfer, and blockwise data compression. Demos are built using loader scripts and demo parts are loaded sequentially. Loader calls are parameterless. Sparkle handles multi-disk trackmos as well. A Win32 tool is provided to edit script files and build demo disks.

MAIN FEATURES

- Tested on 1541-II, 1571, and Ultimate-II+. Passed THCM's rigorous 24-hour test.
- Resident size: $280 bytes including loader, depacker, fallback IRQ ($0180–$02ff), and buffer ($0300–$03ff). Stack is reduced to $0100–017f. The buffer contains preloaded data between loader calls, so it needs to be left untouched.
- Only two bytes are clobbered on the zeropage which can be set from script. Default is $02–$03. OK to use them between loader calls.
- 125-cycle on-the-fly GCR fetch-decode-verify loop tolerating disk rotation speeds of 289-311 rpm across all four speed zones, providing high stability.
- Very simple communication code with reset detection.
- 2Bit+ATN transfer protocol, 72 bycles/block transfer speed. Transfer is freely interruptible.
- Spartan Stepping™ for seamless data transfer across adjacent tracks with zero additional stepper delay.
- Sequential loading only. No random file access.
- Built-in blockwise packer/depacker. The packer compresses demo parts back-to-back. Thus, no partially used blocks are left on the disk.
- Combined fixed-order and out-of-order loading.
- Bus lock. The loader uses $dd00 for communication. The user can freely abuse $dd02 between loader calls as long as $dd00 is left untouched.
- Loading under I/O is supported.

SPARKLE WINDOWS TOOL

The Sparkle Windows tool (written in VB.NET, target .NET Framework 4.5, should work on Windows 7+) features a simple disk monitor and a built-in script editor. D64 and script files can be opened from within the tool or drag-and-dropped to process them. Script files use the .sls (Sparkle Loader Script) extension. Run Sparkle as administrator to associate the .sls file extension with the tool. Once the necessary registry entries are installed you can also build your demo disks by double clicking script files. (This can also be achieved by selecting Sparkle from the Open with… list after double clicking a script file.) Sparkle can also be used as a command line tool (e.g. sparkle mydemo.sls). A simple demo project is provided as an example.

MAIN WINDOW

Sparkle features a simple disk monitor which is the main window of the program showing the hex and PETSCII views of the selected sector and two toolbars. Here you can start a new disk 🅖 , open 📂 , and save 💾 🖫 D64 files, and build demo disks from scripts. On the second toolbar, the leftmost two buttons will show the BAM 🅱 and the first sector of the directory 🅳 . The next four buttons will navigate to the first ⏫ , previous 🔺 , next 🔻 , and last ⏬ track on the disk. The following four buttons will load the first ⏮ , previous ◀ , next ▶ , or last ⏭ sector of the selected track. The next group of six buttons are used to navigate through demo parts, jumping to the first sector of the first part⏮⏮ , the first sector of the previous part ⏪ , the previous sector in the part — , the next sector in the part ✚ , the first sector of the next part ⏩ , and the first sector of the last part⏩⏭ . Finally, on the right side of the toolbar you can see the current or manually select the desired track T: 18 and sector S: 1 . Hovering over a button will show a tooltip with the description of the button's function. In the Hex View panel, you can perform changes in the selected sector.
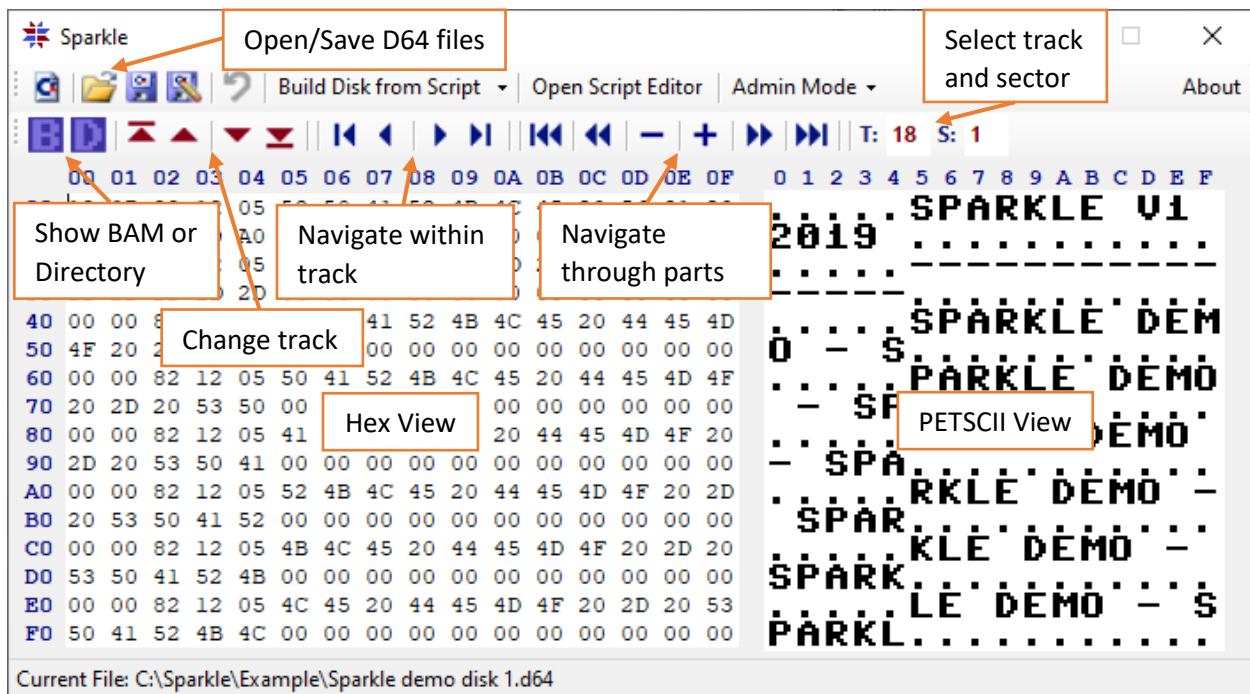
SCRIPT EDITOR

Scripts can be created, loaded and saved in the script editor window where demo information is organized in a treeview structure. Opening the editor will prepopulate the treeview structure with a blank disk or with the previously opened script. Pressing the <Enter> key or double clicking on an item allows editing. Deleting a value will reset it to its default if such exists. Here you can specify where you want to save your demo disk, add a disk header and ID, enter the demo's title (first prg in the directory), the start address (entry point), and you can also add DirArt (16 characters in a row separated by line breaks, saved as a .txt file). Disks, parts, and files can be deleted using the <Delete> key. Demo parts can also be freely rearranged. Scripts can be saved with absolute file paths or relative to the script's path.
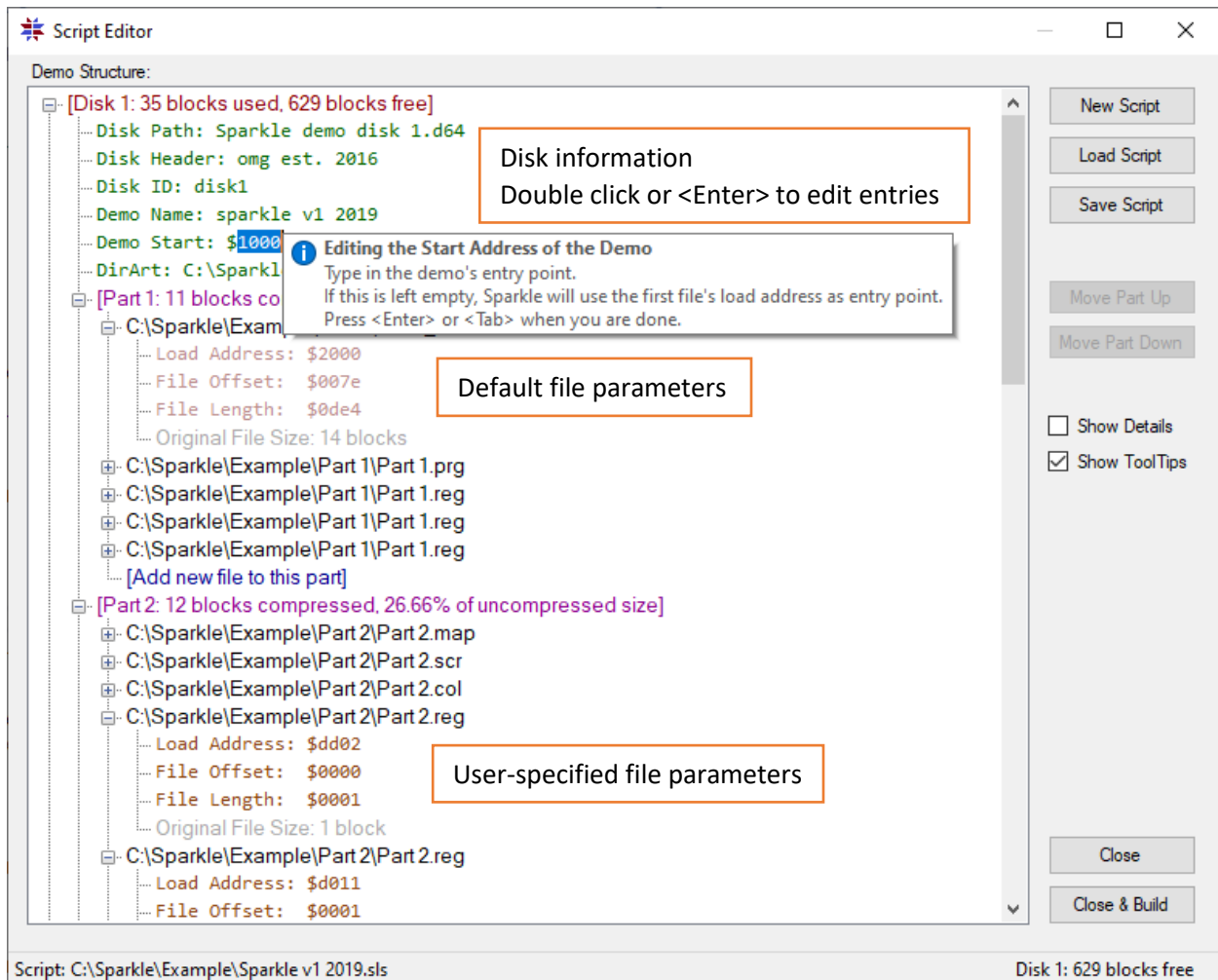
Instead of files, Sparkle works with demo parts. A demo part is the sum of arbitrary files and data segments designated to be loaded during a single loader call. (This may not literally correspond to an actual part in a demo which may require multiple loader calls.) When adding files to a demo part, you can specify what and where you want to load by editing the load address (where the data will go), offset (first byte of data in the file), and length of the desired data segment (number of bytes to be loaded) within the selected file. By default, Sparkle uses the first two bytes of the file as load address, 2 as offset, and (file length-2) as length. If you only specify the load address, then Sparkle will use 0 as offset and the file's length as length. During disk building, Sparkle will only compress the selected segment of the file.

If any part of a data segment overlaps the I/O area ($d000-$dfff) Sparkle will ask you to specify whether the file should be loaded to the I/O (e.g. VIC registers or color RAM) or to the RAM under the I/O. In the latter case Sparkle will mark the filename with an asterisk (*) which indicates that the loader will need to turn I/O off during loading of the specified data segment.

Sparkle sorts files within a part during compression to achieve the best possible compression ratio. Therefore, file order within a part can be random.

3

MANUAL SCRIPT EDITING

Use the following template to manually prepare or edit your scripts:

```
[Sparkle Loader Script]

Path:       your path\your d64 file
Header:     max. 16 characters
ID:         max. 5 characters
Name:       max. 16 characters
Start:      xxxx
DirArt:     your path\your dirart.txt file
ZP:         xx

<Part comment>
File:       your path\your file*  xxxx  yyyy  zzzz
<File comment>
File:

File:
```

The script must start with the file type identifier in brackets and must contain at least one file entry. Everything else is optional and can be omitted. If `Start` is omitted Sparkle will use the load address of the first file entry as start address. Entry types (`Path, Header, ID, Name, Start, DirArt, ZP, File`) and their values are separated by tabs. The loader's zeropage usage (hex byte format, default is `02`) can be set only once, in the first disk's information (see Addendum for a workaround). Lines not recognized as an entry type will be omitted (you may use this feature to add comments to your script). Demo parts are separated by blank lines. Files in a single demo part are in consecutive lines. File paths can be absolute or relative to the script's folder. Disks are divided by adding the phrase `New Disk` in a new line. This is followed by the next disk's information and files.

You can specify three parameters for each file entry, separated by tabs. Parameters are hex numbers in word format (4 digits), prefix is not needed. The first one is the file's load address, the second one is an offset within the file that marks the first byte to be loaded, and the last one is the length of the desired file segment. Parameters can be omitted but each one depends on the one on its left. I.e. you cannot enter the offset without first specifying the load address. Sparkle can handle SID and PRG files so parameters are not needed for these file types unless you want to change them. If all three parameters are omitted then Sparkle will load the file as a PRG file: it will use the file's first 2 bytes to calculate the load address, offset will be 2, and length will be (file length – 2). The only exception is files shorter than 3 bytes for which at least the load address is mandatory. In any case, if only the load address is entered then Sparkle will use 0 for the offset and the file's length as length. If the load address and the offset are given then Sparkle will calculate the length as (file length – offset).

By default, Sparkle turns the BASIC and KERNAL ROMs off during loading but leaves I/O on writing `#$35` to `$01`. If any part of a file is to be loaded to the RAM under the I/O area (`$d000-$dfff`), an asterisk (*) must be added to the end of the file name (see template above). This will tell Sparkle to turn off the I/O area by writing `#$34` to `$01` while unpacking the file. If the * is omitted the file will be loaded to the I/O area (VIC, SID, and CIA registers, color RAM).

RUNTIME CONSIDERATIONS

The C64 resident code and the drive code take 8 blocks on track 18. Thus, you have the entire 664 blocks for your demo and the remaining 10 blocks of track 18 for DirArt. Loading and running any entry from the disk's directory will install the loader which then will automatically load and start the first demo part. The following functions are available:

- Loader call:

```
jsr $0180  //Parameterless
```

   This parameterless subroutine call will load the next part as specified in the script. The loader writes #$35 to $01 at the beginning of every loader call and will return with this value in $01. It does not clobber the I flag. When the loader is called, it first depacks the first partial block (if such exists) of the next part from the buffer before receiving the next block. During depacking I/O may be turned on or off depending on where the data are designated. Once the last part on a disk is loaded the loader moves the read/write head to track 18 and checks the last three bytes of the BAM to determine whether there is a next disk side to be loaded. In case of a multi-disk demo, the next (standard) loader call will instruct the loader to wait for disk flip before the next part is loaded. The loader will reset the drive if there are no more disks.

- Fallback IRQ Installer:

```
jsr $01e0  //X/A = IRQ subroutine vector Lo/Hi
```

   Use it to install a simple fallback IRQ with a music player call or any other function. The IRQ routine is located at $02e1 and the subroutine call initially points at an `rts` instruction. The low and high bytes of the subroutine address need to be in the X and A registers, respectively before the IRQ installer is called. Use `jsr $01e6` if you do not want to change the subroutine vector. The I flag is set during the initialization of the loader and remains set after loading the first part is completed. Calling the IRQ installer does not change the I flag allowing the user to set or clear it at the desired moment.

CAVEATS

VIC bank selection must be done by writing #$3c–#$3f to $dd02. Do not change $dd00.

Loading to pages 1-3 is not recommended as it would overwrite the loader or preloaded data in the buffer. While Sparkle can load files compressed by another packer such as Exomizer, make sure to restore the stack pointer and any other registers and zeropage values as required by the packer before you start your program. Restoring the stack pointer will result in overwriting Sparkle's resident code on the stack, so further loads will not be possible.

Start the Win32 tool from a local or removable drive as it does not seem to work properly from network drives.

DISCLAIMER

Sparkle is a free software and is provided "as is". I am not a professional coder so use it at your own risk and expect bugs. I do not accept responsibility for any omissions, data loss or other damages. Please credit me in your production should you decide to use Sparkle. Feel free to contact me with any questions via PM on CSDB or by emailing to spartaofomgATgmailDOTcom.

Sparta/OMG 2019

ADDENDUM

SCRIPT HACK

The following $46-byte script hack will change the loader's zeropage usage at any time during loading:

1. For the example's sake let's say you want to change zeropage usage from the default `$02/$03` to `$fe/$ff`. Prepare a 4-byte long file, e.g. NewZP.bin with the following content:

    `$fe, $ff,` `$28, $02`

2. Add the following lines to any loader part in your script where the ZP change is needed:

```
File: Your Path\NewZP.bin    00fe  0002  0002
File: Your Path\NewZP.bin    02a8  0000  0001
File: Your Path\NewZP.bin    0228  0000  0001
File: Your Path\NewZP.bin    0237  0001  0001
File: Your Path\NewZP.bin    0294  0000  0001
File: Your Path\NewZP.bin    029a  0001  0001
File: Your Path\NewZP.bin    0296  0000  0001
File: Your Path\NewZP.bin    0246  0000  0001
File: Your Path\NewZP.bin    0267  0000  0001
File: Your Path\NewZP.bin    0269  0000  0001
File: Your Path\NewZP.bin    0271  0001  0001
File: Your Path\NewZP.bin    02b9  0000  0001
File: Your Path\NewZP.bin    02bb  0000  0001
File: Your Path\NewZP.bin    02bf  0001  0001
File: Your Path\NewZP.bin    02c7  0000  0001
File: Your Path\NewZP.bin    02cc  0001  0001
File: Your Path\NewZP.bin    02d7  0000  0001
```

Don't forget to replace the bold text with your desired ZP address. 😊

Note: this hack is intended to be used with the current release. I may not work with future releases.