SPARKLE V1.4

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, files are bundled together and are loaded sequentially in batches. Loader calls are parameterless. Sparkle handles multi-disk trackmos as well. A Windows tool is provided to edit script files and build demo disks. For version history and description of new features please see Addendum 1.

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.
- Three bytes are clobbered on the zeropage which can be set from the script. Default is $02–$04. OK to use them between loader calls.
- 125-cycle on-the-fly GCR fetch-decode-verify loop tolerating disk rotation speeds of at least 284-311 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 bundles of files back-to-back. Thus, no partially used sectors 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, but $dd00 needs to be 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.

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 file bundles, jumping to the first sector of the first bundle ⏮⏮, the first sector of the previous bundle ◀◀, the previous sector in the bundle —, the next sector in the bundle +, the first sector of the next bundle ▶▶, and the first sector of the last bundle ▶▶|. 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 and its shortcut key. In the Hex View panel, you can perform changes in the selected sector.
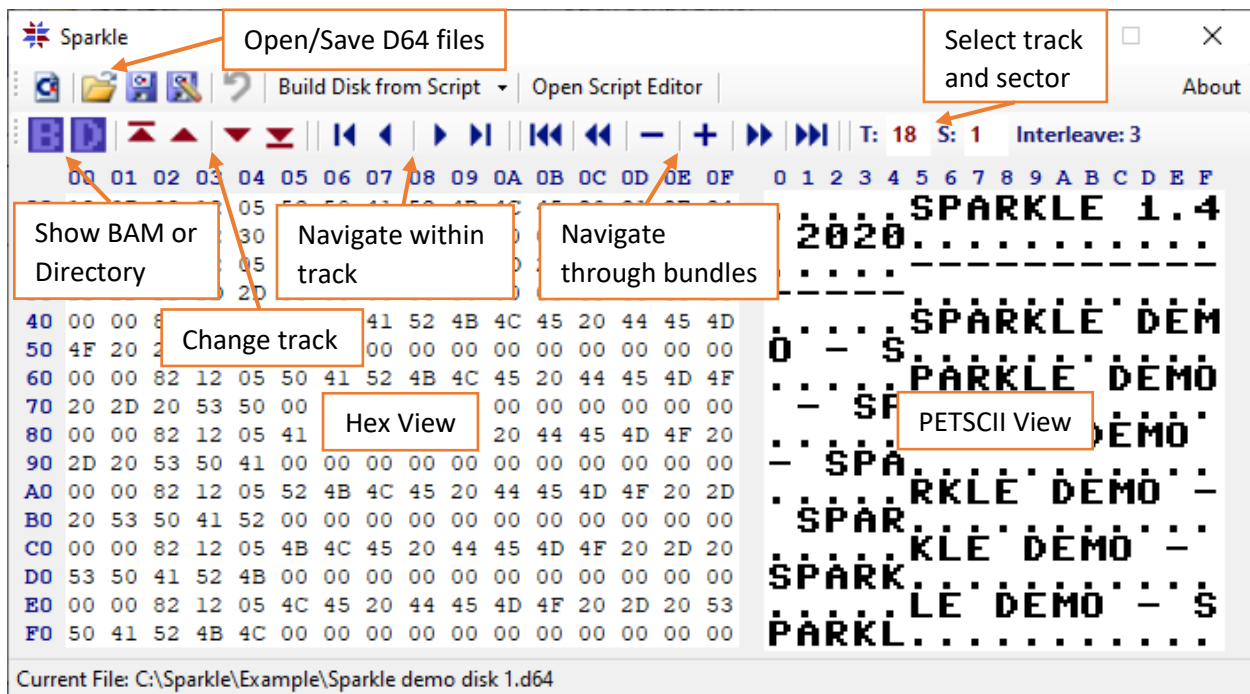
SCRIPT EDITOR

Scripts can be created, loaded and saved in the script editor window where demo information is organized in a treeview structure. Script entries can be edited by double clicking or selecting them and pressing <Enter>. This will initiate editing by selecting the editable text portion of the entry or opening a folder dialog where a file path is needed. Editing entries that accept free text can also be initiated by pressing an alphanumeric (a-z + 0-9) key which will immediately overwrite the entry's last value. Pressing <Escape> will cancel editing and restore the entry's previous value. To finish editing, press <Tab>, <Enter>, <Down> to move selection down or <Up> to move selection up. Entries that have a default value can be reset by deleting their current value and then leaving the entry. DirArt files, disks,

file bundles, demo files, and embedded scripts can be deleted using the <Delete> key. The editor marks default parameters with lighter color then modified ones.

The top line in the editor (This script) shows the current script's path and file name. If you start a new script, it will be updated once you save your script. You can also double-click this entry (or press <Enter>) to load a script. The Add… entry and its three options (New Disk, New Bundle, New Script) are always located at the bottom of the current script and can be used to add any of these three types to the script.
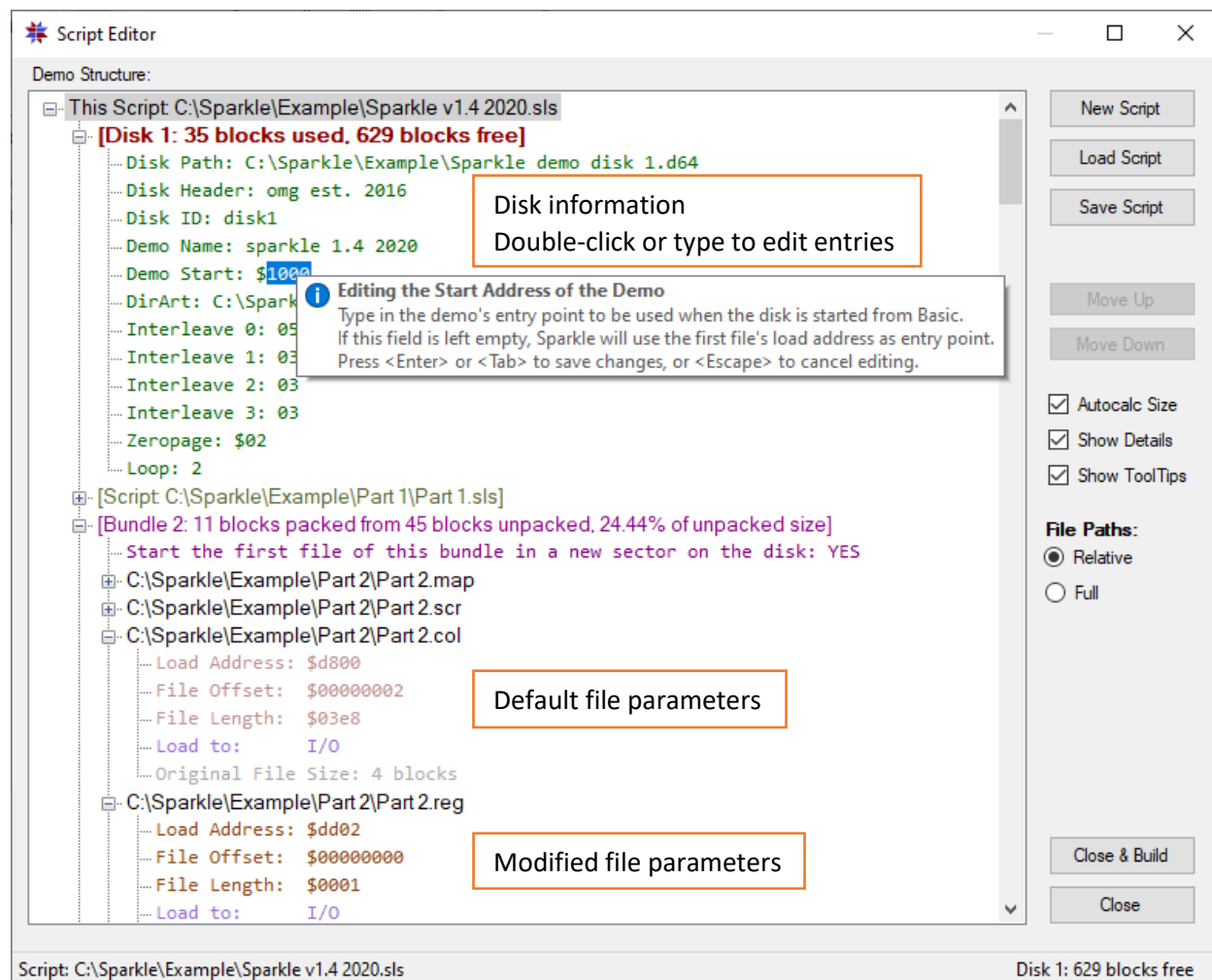
To build a demo disk, open the editor and first add a blank disk structure. In the disk structure you can specify all necessary settings related to this demo disk. Zeropage usage and demo looping are global settings and can be only specified once, in the first disk. The following options are available:

Disk Path: Here you can specify the final D64 file's name and path. Sparkle will use this information when the demo disks are created.

Disk Header: Max. 16 characters that will be used as the disk's header information in the topmost line of the disk's directory structure.

Disk ID: Max. 5 characters that will be located on the right side of the disk header in the directory structure.

Demo Name: Max. 16 characters. This will be your first directory entry on the disk. Sparkle will load the installer when this entry is loaded.

Demo Start: The program entry of the demo when loaded from BASIC. Once the loader is installed, it will load the first file bundle from the disk automatically and then it will jump to this address. If not specified, Sparkle will use the load address of the first file in the first bundle as a start address. (This is the only time the file order matters in a bundle.)

DirArt: Sparkle currently accepts files with line breaks after max. 16 bytes/characters. If a line is longer than 16 characters/bytes then Sparkle will use the first 16 characters/bytes of it as a DirArt entry. There are 10 sectors available on track 18 for DirArt (max 80 entires, including the one specified at the Demo Name entry type). Each DirArt directory entry is executable and will start the demo.

Interleave 0-3: These entries specify the distance in sectors between two consecutive data blocks on the disk. The default values are 4 for tracks 1-17 (speed zone 0), and 3 for tracks 18-24, 25-30, and 31-35 (speed zones 1-3). Sparkle accepts decimal values between 1 and the number of sectors in a track less 1 (20, 18, 17, and 16, respectively) for each speed zone (interleave MOD number of sectors per track in zone cannot be 0). You can change the interleave to optimize Sparkle's performance in each speed zone separately, depending on how much raster time is left for loading. E.g. for speed zone 0, use 4 if you have more than 75% of the raster time available on average for loading from the first 17 tracks, select an interleave of 5 if you have 50-75% of the raster time available, and an interleave of 7 may be appropriate if you only have about 25% of the screen time. Please remember that the interleave can only be specified once for each speed zone on the disk and will be the same for each bundle within the speed zone. Finding the best interleave may need some experimentation. You may want to select one that works best for your most loading time sensitive demo part. Once the loading sequence enters a new speed zone on the disk, a new interleave can be used.

Zeropage: Sparkle uses 3 bytes on the zeropage. The default is $02-$04. If this interferes with your demo (e.g. SID uses the same addresses) then you can change it here. This is a global setting and can only be specified once, in the first disk, and it will be used for the rest of the demo. These zeropage addresses can be used freely between loader calls. Since loading typically runs from the main code, you can also save and restore them from IRQ during loading if needed. Please see Addendum 2 for a workaround if you need to change zeropage usage on the fly.

Loop: A decimal number between 0 and 255 that allows Sparkle to loop the demo once the last bundle on the last disk is loaded. The default value of 0 means no looping. Any other number represents a disk where 1 identifies the first disk of the demo. If a nonzero number is specified, Sparkle will wait for the specified disk to be inserted to continue. If the value of this entry specifies the last disk then it will be loaded continuously, in an endless loop. Loop is a global setting and can only be specified once, in the first disk.

After completing the disk info section, you can start adding files to your disk. Instead of loading files one by one, Sparkle bundles files together and loads them in batches. A file bundle is the sum of arbitrary files and data segments designated to be loaded during a single loader call. You can put any number of files and data segments in a bundle (as long as they do not overlap in the memory) and load them in one call. The more you put in a bundle, the faster loading will be. A bundle may contain files for the next as well as a subsequent demo part. You can also combine files that are loaded in the RAM under the I/O area ($d000-$dfff) with others that are loaded to the I/O at the same memory address.

To add files to your demo disk, first you need to create a new bundle (Add… -> New Bundle). Once this is done, you will see two new entries under the bundle entry. The first one ("Start the first file…") is a bundle-specific setting that determines how Sparkle will add this bundle to the disk. By default, Sparkle saves compressed bundles back-to-back on the disk, leaving no unused space between bundles. I.e. a bundle will first occupy the unused space in the last block of the last bundle before starting a new block in the next sector. During loading, these transitional blocks are left in the loader buffer ($0300-$03ff) between loader calls and the next loader call will first depack the beginning of the next bundle from the buffer before loading the next block. This behavior can be changed by double-clicking the "Start the first file of this bundle in a new sector on the disk" entry which will toggle its value between YES and NO. Changing the value to YES will insert the Align key word in the script file in the line preceding the first file of the bundle which will force Sparkle to leave the last sector of the previous bundle partially unused and start this bundle in a new sector on the disk (similar to how other loaders work). This can be helpful in loading time sensitive demo parts. However, most of the time, this option would just inflate disk usage.

When adding files to a bundle, you can specify what and where you want to load by entering the load address (where the data will go), offset (first byte of the file to be loaded), and length of the desired data segment (number of bytes to be loaded) within the selected file. During disk building, Sparkle will only compress the selected segment of the file. By default, Sparkle assumes that the file is a PRG independently of the file extension and uses the first two bytes of the file as load address, 2 as offset, and (file length-2) as length. The only exception is SID files where file parameters are extracted from the file automatically. If you change the file's load address then Sparkle will presume that the file is NOT a PRG and it will change the offset to 0 and will use the file's full length as length, accordingly. Thus, you will need to make sure that the offset and file length are correct after changing the load address.

If any part of a data segment overlaps the I/O area you can select whether it is to be loaded to the I/O area (e.g. VIC registers, color RAM, etc.) or in the RAM under the I/O area by double-clicking the "Load to" entry under the file. If the file is destined in the RAM under the I/O area 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 examines the I/O status of every data block separately and sets $01 accordingly during decompression.

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

Demo scripts can become very long, so you may want to create shorter scripts, and then add these to your main script. You can add whole disks to your script this way. If a script is saved using relative file paths then the script's path will be used to calculate file destinations when it is embedded in another

script. When Sparkle reaches a script entry during disk building, it will first process its content before continuing with the next entry. Scripts cannot be inserted into an existing file bundle. I.e. files in the embedded script will always start a new file bundle, and won't be added to the current bundle. The content of embedded scripts cannot be edited from their parent script, only separately, the way they were created.

Disks, bundles, and embedded scripts can be freely rearranged. If a disk section does not precede the first bundle then Sparkle will use default disk parameters for the first disk (except if this script is meant to be embedded in another one when the disk section of the parent script will be used). If two disk sections follow each other without any demo files between them then the second one will overwrite the values of the first one during disk building.

Once your script is finished, you can build your demo by clicking the "Close & Build" button. The script remains active until another one is loaded or created. Opening the Editor again will show the active script.

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 as in the directory header
ID:         max. 5 characters as in the directory ID
Name:       max. 16 characters
Start:      xxxx  (program entry when disk started from Basic)
DirArt:     your path\your dirart.txt file
IL0:        n (sector interleave of n>0 for speed zone 0)
ZP:         xx (set once in first disk info section)
Loop:       0 for no looping or disk number 1-255 (set once)

File:       your path\your file*  xxxx  yyyyyyyy   zzzz
File:       your path\your file   xxxx  yyyyyyyy   zzzz

Align
Script:     your path\your script file to be inserted here
```

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, IL0-IL3, ZP, Loop, File, Script`) and their values are separated by tabs. The loader's zeropage usage (hex byte format, default is `02`) and the `Loop` entry type can only be set once, in the first disk's setup (see Addendum 2 for a workaround for ZP usage). File bundles are separated by blank lines. Files in a single bundle are in consecutive lines. File paths can be absolute or relative to the script's folder. After at least one file or script entry is added to the script, you can start a new disk simply by

adding new disk info entry types (`Path`, `Header`, etc.) followed by the next disk's files. See example demo project's script for details.

You can specify three parameters for each file entry, separated by tabs. The first one is the load address of the file segment, the second one is an offset within the original file that marks the first byte to be loaded, and the last one is the length of the file segment. Parameters are hex numbers in word format (max. 4 digits) for the file address and file length, and dword format (max. 8 digits) in the case of the file offset, prefix is not needed. 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. If only the load address is entered then Sparkle will use 0 for the offset and the file's length as length. Sparkle will calculate the length as (original file's length – offset), but max. (`$10000` - load address) if the load address and the offset are given but the length is not. The "`Align`" entry type is used in a new line preceding a bundle or script entry to align it with a new sector on the disk (See the Script Editor section for details).

By default, Sparkle writes `#$35` to `$01` and turns the BASIC and KERNAL ROMs off during loading but leaves I/O on. If any part of a file is to be loaded in the RAM under the I/O area (`$d000-$dfff`), an asterisk (*) must be added to the end of the file name (see template above). This will instruct Sparkle to turn off the I/O area while unpacking the file. If the * is omitted the file will be loaded to the I/O area (VIC, SID, and CIA registers, etc.). A bundle may contain two file segments sharing the same load address if one is destined under I/O and the other one is to be loaded to the I/O area.

Sparkle will ignore any unrecognized entries. Resaving a script from the editor will result in loosing these entries.

RUNTIME CONSIDERATIONS

The installer, C64 resident code, and 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 start the installer which will install the C64 resident code and the drive code, set the I flag, write #$35 to $01, reduce the stack to the lower $80 bytes and set the NMI vector to an `rti` instruction. Then the installer will automatically load the first file bundle. During loading, the value of $01 can be either #$34 or #$35 depending on the destination of the files in the bundle. Once the first bundle is loaded, the loader will restore $01 to #$35 and it will jump to the start address as specified in the script (or to the first byte of the first file in the script if a start address was not specified) without changing the I flag (IRQs remain disabled). The installer and the loader do not alter any other vectors or VIC registers.

From your demo, the following functions are available:

- Loader call:

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

This parameterless subroutine call will load the next bundle of files 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. <u>Loader calls do not clobber the I flag.</u> When the loader is called, it first depacks the first partial block (if such exists) of the next bundle from the buffer before receiving the next block from the disk. The I/O area may be turned on or off during depacking depending on where the data are designated. Once the last bundle 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 (these also control demo looping on the last disk). In case of a multi-disk demo, the next (standard) loader call will instruct the loader to wait for disk flip after which the next file bundle is loaded automatically. 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 `$02dd` 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 bundle 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 `$dd00` while the drive code is active on the drive as this may make the drive believe that the C64 is requesting the next bundle. Once the last bundle is loaded, the drive will reset and `$dd00` can be freely overwritten.

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 loader calls will not be possible.

Start the Windows 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". It is a hobby project of a hobby 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.

Please find the most up-to-date version of Sparkle on GitHub: https://github.com/spartaomg/Sparkle

Sparta/OMG, 2019-2020

ADDENDUM 1

VERSION HISTORY

V1.4

- New feature: Sparkle shows a warning if there are multiple active drives on the serial bus. The demo will continue once all devices but one are turned off. Thanks to Dr. Science/ATL for this feature request.

- Removed optional packer selection. Sparkle now uses an updated version of its former "better" packer with an optimized decompression algorithm.

- Updated GCR loop for increased stability. Sparkle now has a disk rotation speed tolerance of at least 284-311 rpm across all four speed zones.

- Loader "parts" are renamed to file bundles to avoid confusion.

- The disk monitor now highlights the `[00 F8]` file bundle separator sequence.

- Bug fix: the editor did not calculate bundle and disk sizes correctly.

- Other minor bug fixes and improvements.

V1.3

- New feature: script embedding. If your script is very long, you can save part of it in a separate file and then add this to your script using the "`Script:`" entry type followed by <TAB> and the script file's path. When Sparkle reaches a script entry during disk building, it will first process its content before continuing with the next entry. You can even add whole disks to your script this way. Scripts cannot be inserted in an existing file bundle. I.e. files in the embedded script will always start a new file bundle, and won't be added to the current bundle. If relative paths are used, Sparkle will use the path of the embedded script to calculate the path of the files in it.

- New feature: demo looping. Use the "`Loop:`" entry type followed by <TAB> and a decimal value between 0-255 in the *first* disk's info section to determine your demo's behavior once it reaches its end. The default value is 0 which will terminate the demo. A value between 1-255 will be interpreted as a disk number where 1 represents the first demo disk. Once the last bundle on the last disk is loaded Sparkle will wait for this disk to be inserted to continue. If you use the last disk's number then Sparkle will reload the last disk in an endless loop. This entry type can only be used once in a script. If not specified then the default value of 0 will be used.

- New feature: aligning a bundle with a new sector on the disk. By default, Sparkle compresses files back-to-back, not leaving any unused space on the disk. If the length of a bundle changes during demo development, it will affect the compression and distribution of every subsequent bundle on the disk. This may adversely influence the timing of the demo. From the editor, double-click the "Start this bundle

in a new sector on the disk" line under the bundle node to change its value to YES from NO where this type of timing is crucial to force Sparkle to always start the bundle in a new sector on the disk. If you prefer to manually edit your script, use the "`Align`" entry type in a new line preceding a bundle to achieve the same result.

- New feature: custom sector interleave. Sparkle now allows the user the specify the interleave for each speed zone on the disk. The default is 4 for speed zone 0 (tracks 1-17), and 3 for speed zones 1-3 (tracks 18-35). Use `ILn:<TAB>N` in the disk info section in your script where n=0-3 specifies the speed zone and N>0 is a decimal value for the desired interleave to be used during disk building.

- New feature: Sparkle now generates an exit code when running from command line. The exit code is non-0 if there is an error during disk building.

- Improved and updated editor to accommodate the new features. The editor now accepts alphanumeric (a-z, 0-9) characters in addition to <Enter> to start editing an entry. Just press the first character of the new value to overwrite the previous one, or <Enter> if the first character is not alphanumeric. Once you are done editing, press <Enter> or <Down> to step to the next entry, or <Up> to step back to the previous one.

- Updated, more flexible script handling:

> - Sparkle now recognizes both LF and CRLF line endings.

> - The script entry "`New Disk`" is no longer needed to start a new disk during manual script editing. Just add the next disk's info after the last file or script entry. Make sure there is at least one file entry after every disk info section. Otherwise, the next disk info section will overwrite the previous one.

> - Sparkle will skip any unrecognized lines in the script. This can be used to manually comment your script. Manual comments will be ignored in the editor window and will be lost when the script is resaved from the editor.

> - File offset values can be as large as `$ffffffff`. The maximum value of file address and length remains `$ffff`.

- Bug fixes. Thanks to Raistlin/G*P and Visage/Lethargy for testing and feature requests.

V1.2

- Optional better packer. Sparkle now offers two versions of its packer. The original, faster one, and a new, better one. The new packer results in better compression at the expense of slower packing and depacking compared to the original faster but less effective option.

- Minor improvements in the C64 code saving about 10000 cycles on the depacking of a disk side.

- GUI update.

- Bug fixes.

V1.1

- Option to select ZP usage in the script.

- Improved default file parameter handling.

- Minor changes in the editor.

- Bug fixes related to loading under I/O.

V1.0

- Initial release.

ADDENDUM 2

SCRIPT ZEROPAGE HACK

The following script hack will change the loader's zeropage usage during loading. Note: it is intended to be used with the current (V1.4) release. It does not work with previous Sparkle versions and may not work with future versions. If you want to change zeropage usage at the beginning of the disk, use the zeropage entry instead of this script hack as it does not work in very first bundle of a disk.

1. Prepare a 6-byte long file, e.g. NewZP.bin with the following content (where ZP is the new ZP address):

   ZP, ZP+1, ZP+2, $28, $02, $80

2. Replace ZP with the new ZP address then save the text between the lines as ScriptHack.sls in the same folder. <u>Make sure that there is no line break after the last File entry!</u> Embed ScriptHack.sls in your script as the first entry in a bundle where you want the zeropage usage changed.

```
--------------------------------------------------------------------------------
Align
File: NewZP.bin  00ZP  0003
File: NewZP.bin  02ab  0000  0001
File: NewZP.bin  0228  0000  0001
File: NewZP.bin  0237  0001  0001
File: NewZP.bin  0299  0000  0001
File: NewZP.bin  029f  0001  0001
File: NewZP.bin  029b  0000  0001
File: NewZP.bin  0246  0000  0001
File: NewZP.bin  0200  0002  0001
File: NewZP.bin  026f  0002  0001
File: NewZP.bin  0272  0002  0001
File: NewZP.bin  025e  0000  0001
File: NewZP.bin  0260  0000  0001
File: NewZP.bin  0268  0001  0001
File: NewZP.bin  02ba  0000  0001
File: NewZP.bin  02bc  0000  0001
File: NewZP.bin  02c0  0001  0001
File: NewZP.bin  02c8  0000  0001
File: NewZP.bin  02cd  0001  0001
File: NewZP.bin  02d8  0000  0001
File: NewZP.bin  01f2  0002  0001
File: NewZP.bin  0223  0002  0001
--------------------------------------------------------------------------------
```