F256 Binary Files and Headers

A quick look at the ins and outs of 3 types of binary files including .PGX and the $F2, $56 auto-execute variety

On prior pages, we discussed the .PGX format but there are others to mention. Differences aside, most have two things in common: a) somebody though it a good idea to define a format and b) it wasn't just an idea; one or more people invested effort to create a loader, editor, or attribute viewer. With sound design and some amount of good fortune, formats and standards catch on; in rare cases, they form the basis of a solid ecosystem that contributes to make a good platform, great.

The most simple header I'm aware of, was the Commodore .PRG format. Kernal LOAD and SAVE vectors counted on it to inform the loading of binary and BASIC programs, disk directories, and even app data (e.g. SpeedScript documents). It did so with only two bytes of meta-data, representing the load address in low-byte/high-byte format. Unfortunately, nothing within this thin spec identified the execution address, and if you are familiar with the C64 and its ancestors, you'll probably know that the characters 'P', 'R', 'G' are not part of the filename at all; they manifested from byte 0 of a file's directory entry, occupying a column in the '$' output along with SEQ and REL. In the old days, if you didn't know where or how to load something, detective work was necessary. Proprietary loaders adorned nearly every multi-file program, especially commercial titles. Thankfully, standards have evolved since the '80s and our lives are improved in this regard.

Foenix platforms do not have an 8.3 filename spec or a system extension registry. Extensions are merely a few courteous characters at the end of a filename. In this one-pager, we will profile a few of these formats. Ultimately, the job is the same; to load (one or sometimes, many) files into memory where they belong so a program or the user can use them.

	.PGX	$F2, $56 auto-exec	Generic binary
Byte format	Byte 0 signature: "PGX" Byte 3 $03 CPU type Byte 4 $00 $40 Destination addr. Byte 6 $-- $-- (unused) Byte 8- $20 $21 $12 (data)	Byte 0 signature: $F2, $56 Byte 2 size of program in 8k blocks Byte 3 starting slot Bytes 4-5 start address (to be exec'd) Bytes 6-9 reserved Bytes 10- zero-terminated program name Byte x- (data)	*None* - these binaries are unstructured, but common convention on 6502 platforms has been to align the desired start address to the reset vector ($fffc). In some cases, developers have created their own formats. We'll point to a few app specific use-cases below.
Description	PGX files (see pg. 3) contain identifying information and a load address and can start a program composed of a single segment. PGZ files are similar, but have provisions for multiple segments, loaded across memory (see links below).	Auto-exec binaries, marked by $F2, $56 (aka F256), in bytes 0 and 1 are scanned for by the kernel at start-up. The MicroKernel code that manages startup is MMU 'slot' aware.	Loosely, any file type may be called *binary*. More commonly, the name refers to non-ASCII data that is largely unprintable. A char set 'font' file is an example of one, as is machine code or sprite graphic data loaded by SuperBASIC, noted below.

How to load and execute	Upon load, data is placed in memory at byte offset $08 (the header is processed, but not stored to memory).
	When the application ends (with an `rts`), the program that initiated the load may either resume control, or reset the system.
	Here is the "from SuperBASIC" form, discussed on pg. 3:
	`/- {program}.PGX` (convention; mind the gap; a space is required)
		At detection (during boot or reset), *x* number of blocks (byte 2), consecutive from the signature block, are moved to slot *y* (byte 3) and execution begins from the low-byte/high-byte address in bytes 4 & 5.
		Combined with the ability to flash blocks to cartridge or onboard flash (via the F256 Uploader) *and* use the MicroKernel DOS `lsf` command (see figure 4a), this capability is of great use to developers and users interested in customizing their system startup and onboard toolset.
			Absent identifying information indicating how or where to load or execute, the parent or calling program must load the data into memory at a prescribed address. Within SuperBASIC, this can be done as follows:
			`BLOAD "{filename}", {[$]addr}`
			… and if code, it can be executed using:
			`CALL {[$]addr}`
			There are several examples present in the *Foenix F256 Graphics Toolkit* by Ernesto Contreras (see link below).

.asm directives	`* = $4000`
	
	`.text "PGX"`
	`.byte $03`
	`.word keys.cmd`
	`.word 0`
		`* = $4000`
		
		`.text $f2,$56`
		`.byte 1`
		`.byte 2`
		`.word keys.cmd`
		`.word 0`
		`.text "sak - standalone keys",$0`
			Not applicable

See also	https://wiki.c256foenix.com/index.php?title=Executable_binary_file	https://github.com/ghackwrench/F256_Jr_Kernel_DOS/blob/main/kernel/README.md (see the "**Startup**" section)	http://apps.emwhite.org/shared-files/770/?F256-GraphicToolkit-11.zip&download=1 See line 3500 onwards in the Foenix Sprite Editor "`spreditjr.bas`" for the BLOAD sprite load code. See line 3650 onwards in the Foenix Font Editor "`fontjr.bas`" charset load / relocate code.

https://github.com/FoenixRetro/Documentation/blob/main/f256/programming-file-formats.md

(applies to both)