FYSOS Specification
Revision 0.85
24 September 2017

1. Introduction
This specification defines the requirements for the FYSOS operating system. It is for the Intel x86 processor, running in 32-bit protected mode, using little-endian addressing.

More information can be found at http://www.fysnet.net/fysos.htm

This specification is so that, when followed, any boot loader can be used to load and boot to the kernel as well as writing drivers for the kernel.

This specification specifies the requirements for a compatible boot, loader, and kernel members of the system, as well as driver compatibility requirements.

1.1. Terms
Boot Code	The code used to load the Loader.
Boot Loader	Same as Boot above.
Loader	The code used to load the Kernel and other files.
Kernel	The code used to initialize and use the system.

1.1.1. Word Sizes
1) A byte is 8-bits in size
2) A word is 16-bits in size
3) A dword is 32-bits in size
4) A qword is 64-bits in size

1.1.2. Fixed Addresses
There are three (3) fixed physical addresses that must be used within this specification.

Name	Physical location	Description
BOOT_ADDR	0x07C00	Physical start of boot code
LDR_ADDR	0x0C000 -> 0x6FC00	Physical start of boot code
STACK_BASE	0x01000000	Physical start of Kernel Stack

The BOOT_ADDR is fixed via the BIOS. The BIOS will load the single boot sector to this address. The second is the LDR_ADDR address. This is the address the boot code must load the loader.sys file to. The boot code must randomly choose an address between the two values above, with this address on a 200h address boundary. Refer to section 2.4.

1.2. Legacy or UEFI BIOS

When discussing the Boot Loader, Section 2 is used for Legacy BIOS only while Section 3 describes both Legacy BIOS and UEFI systems.

1.3. Processor Restrictions

It is required that any code checks for any processor instructions and/or capabilities before using said instructions or capabilities except that of 16-bit processors. For example, any code may assume and use 16-bit code up to and including the 80x286 processor without first detecting for that processor. However, any processor and capability after the 80x286 must be detected before used.

1.4 Independence

Each item, Boot Loader, Loader, and Kernel, must be independent of each other, except when specified. For example, the Boot Loader must be independent of the Loader and Kernel, other than the information shown in Section 2.1.

Any Boot Loader, when following this specification, can be used to find and load the Loader as well as any Loader can be assumed to be safely loaded by any Boot Loader, as long as this specification is followed.

1.5 Partition Schemes

It is outside this specification on what partition scheme is used, though the only requirement of the partition scheme is to find the active bootable partition, aka volume, load the first sector of that volume, set specified registers as the BIOS or other firmware designates, and jump to loaded code.

As of this writing, the FYSOS system understands the following partitioning schemes:

None	Volume occupies all of media or may occupy first part of media including first sector of media. Any sectors after this volume is inaccessible.
MBR	Legacy Master Boot Record containing one to four partition entries. One or more of these entries may contain an extended partition type pointing to one or more entries.
eMBR	Extended Master Boot Record. A partitioning scheme allowing large volumes. Described at: http://www.fysnet.net/fysos_embr.htm
EFI GPT	Intel's (u)EFI GUID Partition Table partitioning scheme allowing large volumes

2. Boot Loader

The Boot Loader is the code that gets executed by the BIOS after the POST, i.e.: after the BIOS has done its part at power up, then loaded the first sector of the first bootable device to BOOT_ADDR. This code has been loaded from the disk by the BIOS.

It is the job of the Boot code to:

1) boot the volume
2) parse any partitions, finding the active partition

3) find the loader file and load it to physical address LDR_ADDR. Refer to section 2.4.
4) jump to that address

After the Loader is loaded, the boot loader must do the following before jumping to the loader:
1) the stack must be at least 256 16-bit words. The loader is free to create a new stack.
2) must pass a specific structure of data in EBX, described later.
3) have loaded the complete loader.sys file from the partition
4) start execution at physical address LDR_ADDR

This specification does not limit the type of partitioning of the disk. The disk may be one single large continuous string of sectors or divided into multiple groups of continuous sectors. The boot loader may use as much of the disk as needed, when the file system allows, to complete the tasks listed in this section.

2.1. Boot Data Structure

Before the boot loader jumps to physical address LDR_ADDR, it must fill a memory block with specific data and pass the physical address to the loader in the EBX register. The address of this data block can be anywhere accessible via 16-bit segmented addressing. The boot code must leave this physical address in the EBX register as it jumps to LDR_ADDR.

This data structure has the following format:

```
S_BOOT_DATA struct
  signature    dword  ; sig used for finding booted from partition
  base_lba     qword  ; base lba of partition
  loader_base  dword  ; physical address of loader.sys
  file_system  byte   ; filesystem number
  drive        byte   ; BIOS drive number
  reserved     dup 30 ; padding/reserved (30 bytes)
S_BOOT_DATA ends
```

2.1.1. Signature

So that the Kernel may detect the partition it was booted from, the Boot code needs to acquire and/or create a signature value uniquely identifying this partition. This signature is created depending on the File System it booted from. For example, if booted from a FAT file system, this signature can be the signature from the File System's BPB. This signature is usually created and written at volume build time. The creation of this signature is not part of this specification, it only needs to be unique to all other volumes known at boot time. Refer also to section 2.3.

2.1.2. Base LBA

The Loader, and later the Kernel, will need to know a few things about the booted from device. For example, the boot code will be the only member that can acquire the base LBA address of the booted partition. The Loader has no way of knowing where the base of this volume is. Therefore,

the boot code must pass this address to the Loader, which will later pass it to the Kernel. Refer also to section 2.3.

2.1.3. Loader Base

This is the physical base address of the loader, which the boot code has loaded this loader to. This is so the loader may know where it is in memory.

2.1.4. File System

So that the Loader does not have to detect the file system is resides on, and since the boot code already knows the file system it is booting, the boot code must pass a file system signature value to the Loader, indicating the parent file system. The table below shows the valid values used.

Table 2.1: File System Signatures

Value	Description
0	Unused. Non-specified file system used.
1	Lean FS http://freedos-32.sourceforge.net/lean/
2	Ext2 Linux Ext2 File System
3	SFS -- Simple File System v1.10 https://github.com/fysnet/mksfs/blob/master/sfs.pdf
11	ExFAT -- Extended FAT https://en.wikipedia.org/wiki/ExFAT
12	FAT 12 http://www.fysnet.net/docs/fatgen103.pdf
16	FAT 16 http://www.fysnet.net/docs/fatgen103.pdf
22	FYS FS http://www.fysnet.net/fysfs.htm
32	FAT 32 http://www.fysnet.net/docs/fatgen103.pdf

Any other value is considered in error. Refer to section 3.

2.1.5. BIOS Disk Value

The BIOS passes the DL value, the drive value used by the BIOS disk services, to the boot code. Since the Loader will also use these services, the Boot code must also pass it to the Loader.

2.1.6 Reserved

The remaining field(s) in the Boot Data Structure are reserved and should be zeros.

2.2. Different Source Boot files

The developer may have one or more source files used to create the binary for the boot code of this specification. For example, the media may be formatted as the original MBR partitioning or to the new GPT partitioning. It is not specified in this specification which partitioning scheme is used or how to code for it. The code simply needs to complete the tasks for this section. Therefore, the developer may create numerous binary files from numerous source files, each binary file ready for any specific type of partitioning scheme. Then once the scheme is chosen, a specific binary file may be written to the disk at that location after any at-write-time modifications that need to be made.

2.3 Signature and LBA in Boot Sector

No matter the boot code used, nor the file system used, all FYSOS compliant boot sectors must contain the following structure at address 0x1F2 relative to the start of the first sector of the boot code.

```
S_SIG_DATA struct
  signature    dword  ; sig used for finding booted from partition
  base_lba     qword  ; base lba of partition
  BIOS_sig     word   ; Boot sector signature (0xAA55)
S_SIG_DATA ends
```

The signature member is a unique signature to this volume. The calculation of this signature is outside this specification, but it must be unique to all volumes on this system. The base_lba member must be the base sector address of this sector, the first sector of this volume. These two members are then copied to the same named members of the structure explained in section 2.1. The last member is the BIOS required two-byte signature of 0x55 0xAA, also written as 0xAA55.

2.4 Loader Address

So that the Loader can be malware resistant, the Loader needs to be relocatable to any address from 0x0C000 to 0x6FC00, inclusive, and must be on a 512-byte boundary. The Boot code must randomly choose a location from this address area to place the Loader code. The Loader code must then be coded so that it can be loaded to any of said addresses. This is to keep away from fixed addresses, helping to keep malware from taking over. The Boot code then passes this address to the Loader code via technique from section 2.1.

3. Active Booted Partition

This specification does not define the file system used and can be of any file system as long as both the Boot code and Loader code are capable of loading from said file system. See section 2.1.4 for currently supported file systems. If one of these file systems is used, the Boot code must pass the associated value to the Loader code. If a non-specified file system is used, the Boot code should pass a zero in the associated field described in section 2.1, and the Loader should already know the file system used, or detect it itself.

3.1 Loader File

The current Loader file used with FYSOS is coded to use all file systems specified in section 2.1.4. However, it is not required by this specification for a Loader to support all file systems. A Loader may only support one file system and a separate Loader file may be written for any other or multiple other file systems. It is only required that the Loader file know the file system it currently resides on.

3.2. Partition's File System

It is up to the Boot and Loader code to have any required file system functionality to parse the file system to find and load the required files. The Boot and Loader must be independent of each other when parsing the file system. i.e.: the Loader must not rely on any code or data already loaded by the Boot code. The Loader must create and parse the parent file system on its own, only relying on the file system signature value passed from the boot code.

The loader file must be named "LOADER.SYS" and must be case insensitive. If the file system allows for case-sensitive file names, the loader will use the first found file entry that matches this name using case insensitive matching. It is outside this specification on how the Boot code finds the first matching filename.

When using a Legacy BIOS environment, the loader file must be located and accessible via the Legacy BIOS disk-read services. For example, the Legacy BIOS might only use 32-bit LBA's. Therefore, no matter the BIOS used, the loader file must be located no more than a 32-bit LBA address from the first of the media. Note that this requirement is not a 32-bit address from the start of the volume, but an address from the start of the media.

If the file system used allows folders/directories, the Loader file must be within one of the following named folders. The Boot code must search in these folders using this order.

If a Legacy BIOS is used:

/ (root)
/boot
/system/boot

If a UEFI BIOS is used:

/efi/boot
/ (root)

If the loader file is not found, the Boot code must give an error and halt execution.

4. Loader

At this point, the loader may assume that the loader file has been completely loaded to LDR_ADDR and that the EBX register points to a valid Boot Data Structure as described in Section 2.1 of this specification. Other than this Boot Data Structure or anything else described up to this point, the Loader must not rely upon any previous state of the processor and or memory.

The Loader must start in 16-bit real mode assuming an 80x286 processor with a minimal stack frame.

It is the job of the Loader to do the following using a close resemblance and order as listed below:

1) Check for and detect highest required processor state
2) Create Loader transfer area and mark BIOS type used (Refer to section 4.2)
3) Pre-initialize any data or code
4) Create an address space allowing access to all physical memory
5) Retrieve the Memory Allocation data from the BIOS/firmware
6) Detect a PCI Bus
7) Load the listed system files from the parent file system
8) Retrieve or create a dummy Int1E data block
9) Reset the BIOS Disk Services (Legacy only)
10) Get BIOS equipment list (Legacy only)
11) Get Disk Drive Parameters
12) Get BIOS time of day
13) Detect an APM BIOS (Legacy only)
14) Retrieve allowed video modes, setting a mode
15) Move Loader transfer data block to specified location just before the kernel
16) Create Kernel's Address space, moving to 32-bit protected mode
17) Create Kernel's Stack
18) Jump to Kernel

4.1. Detect Processor State

The Loader code must detect the capabilities of the processor before using any said capabilities as described in Section 1.3 of this specification. The Loader must check for the following requirements used later by the Kernel. If any of the following requirements are not met, the Loader is to give an error and halt execution.

1) A 32-bit protected mode environment capable of a paging memory system
2) The CPUID instruction
3) The RDTSC instruction

Other than the above requirements, the Loader needs only to detect what it plans to use.

4.2. Loader Transfer Block

Since much of the necessary data and information from the BIOS must be retrieved before moving to the Kernel, the loader code must set up a data block to hold said information. This data block

must use the exact format described later and just before Kernel execution, placed at physical address (KERNEL ADDRESS + 0x400 - size of this block). For now, since the KERNEL ADDRESS value is not yet known, use any address within the 16-bit address space.
This Loader Transfer Block must use the exact format and size listed below. See the remaining sub-sections for descriptions on the format of these items.

Table 4.1: Loader Transfer Block

Name	Size	Type	Description
Magic #0	4	Both	First Magic Number (0x464F5245) (‘FORE’)
GDT Entry Offset	6	Both	GDT data used when loading the Kernel’s GDT
IDT Entry Offset	6	Both	IDT data used when loading the Kernel’s IDT
BIOS Type	4	Both	BIOS type used: ‘BIOS’ or ‘UEFI’
UEFI Image Handle	4	EFI	Used for accessing the UEFI from the Kernel
UEFI System Table	4	EFI	Used for accessing the UEFI from the Kernel
Boot Data Structure	48	Both	The Data structure retrieved from the Boot (Section 2.1)
Reserved	32		Reserved for future use
Magic #1	4	Both	Second Magic Number (0x56455259) (‘VERY')
BIOS PCI Data	8	Both	PCI information from the BIOS
Original INT 1E Addr	4	LO	Address to the original Int 1E data block (segmented)
Original INT 1E	11	LO	Data retrieved from address above
Time	14	Both	BIOS Time of Day
APM	44	LO	BIOS Advanced Power Management
Reserved	3		Reserved / used for alignment
BIOS Equipment	2	LO	Int 11h data from 0x00410
Keyboard Status	1	LO	Keyboard status from 0x00417
Magic #2	4	Both	Third Magic Number (0x4F554E47) ('OUNG')
Memory Allocation	1,356	Both	Memory Allocation Description from BIOS
A20 Activation	1	Both	A20 Activation Technique used
Text Only Flag	1	LO	Use Text only (Legacy only) Must be zero for EFI.
Video Mode Count	2	Both	Count of Video Mode Blocks below
Current Mode Index	2	Both	Current Video Mode Block Index
Video Mode Block	768	Both	Video Mode Block (32 entries)
Reserved	28		Reserved / used for alignment
Drive Parameters	960	Both	Drive Parameter Table Entries (10 entries)
Reserved	1,795		Reserved for future use
Magic #3	4	Both	Forth Magic Number (0x534F4654) ('SOFT')
Total Size	5,120	Key: LO = Legacy BIOS Only (cleared for EFI BIOS) EFI = EFI BIOS Only (cleared for Legacy BIOS) Both = Required for Both	

As specified in Section 4.15, once initialized and filled, this data block is moved to just before the kernel.

4.3. Pre-initialize

You must pre-initialize any data used by the Loader that will be passed to the Kernel. For example, the data block shown in Section 4.2 must initially be set to all zeros. Also, each system file loaded will be CRC32 checked, therefore, you must initialize all CRC32 routines. See Section 6.1.4.

The remaining of this sub-section describes the items within the table shown in Section 4.2

4.3.1. GDT Entry Offset

This field must contain the count and physical address of the Kernel's GDT as defined in Section 4.16.

```
size         word   ; (Count of entries * 8) - 1
address     dword   ; physical address of GDT
```

A valid GDT must be created before Kernel execution as shown in Section 4.16.

4.3.2. IDT Entry Offset

This field must contain the count and physical address of the Kernel's IDT as defined in Section 4.16.

```
size         word   ; (Count of entries * 8) - 1
address     dword   ; physical address of IDT
```

A valid IDT must be created before Kernel execution as shown in Section 4.16.

4.3.3. BIOS Type

The field must contain one of two values depending on the type of BIOS found. If a Legacy BIOS was found, this field contains the value 'BIOS', the 'S' written as the low-order byte, 'O' next, etc., producing the 32-bit value of 0x42494F53. If a UEFI BIOS was found, this field contains the value 'UEFI', the 'I' written as the low-order byte, etc., producing the 32-bit value of 0x55454649.

4.3.4. UEFI Image Handle

This is the 32-bit handle value used when accessing the UEFI services. This field must be zero when a Legacy BIOS type is found.

4.3.5. UEFI System Table

This is the 32-bit physical address of the UEFI System Table and is used when accessing the UEFI services. This field must be zero when a Legacy BIOS type is found.

4.3.6. Boot Data Structure

This field is updated with the passed data of the same structure from the Boot code as shown in Section 2.1.

4.3.7. Reserved
Any field marked Reserved in Table 4.1 must be pre-initialized to zeros before passing to the Kernel.

4.3.8. BIOS PCI Data
This field is initialized with the following members retrieve from the BIOS PCI Services.

```
sig         dword   ; BIOS Signature
flags       byte    ; flags
major       byte    ; Major version of PCI BIOS
minor       byte    ; Minor version of PCI BIOS
last        byte    ; highest number of bus found
```

This field must be zero when a UEFI BIOS type is found.

4.3.9. Orignal INT 1E Address
This is the segmented address of the original BIOS Interrupt 1Eh Vector pointing to the BIOS Floppy Disk structure. The format of this field is defined by the Legacy BIOS Floppy Diskette Parameter Table. This field must be zero when a UEFI BIOS type is found.

4.3.10. Orignal INT 1E Contents
If the field from Section 4.3.9 is valid, this field must hold the original values from the segmented address found as described in Section 4.3.9. This field must be cleared to zeros when a UEFI BIOS type is found.

4.3.11. Time of Day
This field must be filled with the current Time of Day as retrieved from the BIOS and has the following format.

```
year        word    ; current year in YYYY format
month       byte    ; current month in MM format (one based)
day         byte    ; current month in DD format (one based)
hour        byte    ; current hour in HH 24-hour format (zero based)
min         byte    ; current minute in MM format (one based)
sec         byte    ; current second in SS format (one based)
jiffy       byte    ; current jiffy in JJJ format (zero based)
msec        word    ; current millisecond (zero based)
dl_savings  byte    ; 0 = Daylight-savings not in use
weekday     byte    ; current weekday index (zero based)
yearday     word    ; current day of year (zero based)
```

The `jiffy` field and any field after may contain zeros, though all fields before must be properly initialized to the current time and date.

4.3.12. BIOS Advanced Power Management

This field contains the information from the Legacy BIOS' APM services.

```
present     byte    ; non-zero if the APM BIOS was found
initialize  byte    ; non-zero if the APM BIOS was initialized
maj_ver     byte    ; major version found
min_ver     byte    ; minor version found
flags       word    ; flags (see APM BIOS Specification)
batteries   byte    ; (see APM BIOS Specification)
vap_flags   word    ; (see APM BIOS Specification)
error_code  word    ; (see APM BIOS Specification)
reserved    5 dup   ; reserved
cs_seg32    word    ; (see APM BIOS Specification)
entry_point dword   ; (see APM BIOS Specification)
cs_seg16    word    ; (see APM BIOS Specification)
gdt_idx_16  dword   ; (see APM BIOS Specification)
gdt_idx_32  dword   ; (see APM BIOS Specification)
gdt_idx_ds  dword   ; (see APM BIOS Specification)
ds_segment  word    ; (see APM BIOS Specification)
cs_len32    word    ; (see APM BIOS Specification)
cs_len16    word    ; (see APM BIOS Specification)
ds_length   word    ; (see APM BIOS Specification)
```

This field must be cleared to zeros when a UEFI BIOS type is found.

4.3.13. Reserved

Any field marked Reserved in Table 4.1 must be pre-initialized to zeros before passing to the Kernel.

4.3.14. BIOS Equipment List

This field is read from the Legacy BIOS at physical address 0x00410 as a 16-bit word. This field must be cleared to zeros when a UEFI BIOS type is found.

4.3.15. Keyboard Status

This field is read from the Legacy BIOS at physical address 0x00417 as an 8-bit byte. This field must be cleared to zero when a UEFI BIOS type is found.

4.3.16. Memory Allocation

This structure is filled with the current memory allocation used and returned by the BIOS and has the following format.

```
type_used  word     ; type of BIOS service used to get information
size       qword    ; Total size of memory in K's
```

```
count      word     ; count of entries filled
 .base     qword    ; base physical address of block
 .size     qword    ; size in bytes of this block
 .type     dword    ; type of memory found
 .attrib   qword    ; See UEFI v2.5, page 157
```

The `type_used` field is set to one of the following values, while all other values are invalid.

Table 4.3.16.1: Memory Allocation Type

Value	Type	Description
0	Both	Not found. No memory structure given
1	LO	Service E820h used
2	LO	Service E801h used
3	LO	Service 88h used
4	LO	CMOS used
16	UEFI	UEFI v2.5 Memory Services used

The type field is set to the type of memory entry found and differs whether a Legacy BIOS or UEFI BIOS was found.

Table 4.3.16.2: Memory Allocation Entry Type: Legacy BIOS

Value	Type	Description
1	LO	Memory Available to OS
2	LO	Reserved / System ROM / Mem-mapped Device
3	LO	ACPI Reclaim Memory
4	LO	ACPI NVS Memory

Table 4.3.16.3: Memory Allocation Entry Type: UEFI BIOS

Value	Type	Description
1	UEFI	Loader Code
2	UEFI	Loader Data
3	UEFI	Boot Services Code
4	UEFI	Boot Services Data
5	UEFI	Runtime Services Code
6	UEFI	Runtime Services Data
7	UEFI	Conventional Memory (Available to OS)
8	UEFI	Unusuable Memory
9	UEFI	ACPI Reclaim Memory
10	UEFI	ACPI NVS Memory
11	UEFI	Mem-mapped Device
12	UEFI	Mem-mapped Device
13	UEFI	PAL Code

See the UEFI Specifications, version 2.5, for more information on Table 4.3.16.3 values.

Up to 48 usable entries must be allowed.

4.3.17. A20 Activation Technique

This field is set to the technique value used to activate the A20 line.

Table 4.3.17.1: A20 Activation Technique

Value	Type	Description
0	Both	Not set
1	LO	Keyboard Method
2	LO	Fast A20
3	LO	Keyboard Method (DFh)
4	LO	Brute Force Method 1
5	LO	BIOS A20 Service
6	LO	HP Vectra
7 - 15	Both	Reserved
16	UEFI	UEFI already set it for us

4.3.18. Video Mode Count

This field must contain the count of video entries used as described in Section 4.3.20.

4.3.19. Video Mode Current Index

This field must contain the zero-based index into the video entries used as described in Section 4.3.20 of the current video mode selected.

4.3.20. Video Modes

This field contains the found video modes supported by both the video card and the monitor as returned by the BIOS and contains up to 32 entries as shown below.

```
.lfb        dword    ; physical address of Linear Frame Buffer
.xres       word     ; horizontal resolution (zero based)
.yres       word     ; vertical resolution (zero based)
.bytes_line word     ; bytes per scan line
.bits_pixel word     ; bits per pixel
.mem_model  byte     ; memory model (type of pixel used)
.red        word     ; bit mask: hi byte = bit starting position
                     ;           low byte = count of bits used
.green      word     ; bit mask: hi byte = bit starting position
                     ;           low byte = count of bits used
.blue       word     ; bit mask: hi byte = bit starting position
                     ;           low byte = count of bits used
.reserved   5 dup    ; five reserved bytes
```

The Memory Model field is used to indicate how the pixels are written to memory. Some memory models assume fixed bit fields, so the red, green, and blue fields will be zeros.

Table 4.3.20.1: Video Mode: Memory Model

Value	Description
4	Use bits per pixel value, then assume fixed masks
8	If bits per pixel value is 8, use 0x0503, 0x0203, and 0x0002 for the red, green, and blue fields respectively, else must gather info from respected bit-mask fields.

When Memory Model 4 is used the following pixel types are assumed.

Table 4.3.20.2: Assumed Pixel Type

Bits/Pixel	Description
8	An 8-bit palette is used, created by the system.
15	xRRRRRGGGGGBBBBB
16	RRRRRGGGGGGBBBBB
24	RRRRRRRRGGGGGGGGBBBBBBBB
32	xxxxxxxxRRRRRRRRGGGGGGGGBBBBBBBB

4.3.21. Reserved

Any field marked Reserved in Table 4.1 must be pre-initialized to zeros before passing to the Kernel.

4.3.22. Drive Parameters

This field is initialized with up to ten physical drives as reported by the BIOS. Each entry is shown below.

```
.drv_num    byte     ; Drive number: 0x80, 0x81, etc. (0 = not used)
.extended   byte     ; non-zero = service 48h was used (Legacy only)

; if ‘extended’ above is non-zero
  .ret_size   word     ; size of data (See Legacy BIOS service)
  .info_flags word     ;
  .cylinders  dword    ; cylinders found
  .heads      dword    ; heads found
  .spt        dword    ; sectors per track
  .sectors    qword    ; total sectors found
  .bytes_sect word     ; sectors size

; if ‘extended’ above is zero
  .reserved  26 dup    ; reserved
```

```
.EDD_address dword     ; segmented pointer to EDD info

; version 3.0 data from Legacy BIOS Disk Service
; See Legacy BIOS Service for more information)

.reserved    12 dup    ; padding to 80 bytes

.EDD_info    16 dup    ; EDD information (See Legacy BIOS Service)
```

Each entry must be 96 bytes and must allow up to 10 entries. The `drv_num` member must be valid and may contain zero indicating the entry is not used. All other members are optional.

4.3.23. Reserved for Future Use

Any field marked Reserved in Table 4.1 must be pre-initialized to zeros before passing to the Kernel.

4.4. Loader Address Space Initialization

So that the Loader has a valid address space, and can access memory above the 16-bit addressing limit, you must set up unreal mode and have at least one segment selector set to allow access to all of physical memory.

This is usually done with the FS and GS segment registers, as the normal Legacy BIOS will not use these registers. As for the UEFI BIOS, you should already have a flat address space but do not assume so.

It is not in this specification how you access memory above the 1 Meg mark, but it is required that you do since the necessary kernel files may be and most likely are above that mark.

4.5. Memory Allocation Data

You must initialize the Memory Allocation field in the Loader Transfer Block as described in Section 4.3.16. To do this using the Legacy BIOS, start with service E820h and if not successful, continue to use services from Table 4.3.16.1 until successful or mark as empty. To do this using the UEFI BIOS, call the Boot Services Memory Map service.

4.6. The PCI BIOS Data

Fill in the PCI BIOS Data block as shown in Section 4.3.8. This information must be zero when a UEFI BIOS type is found.

4.7. Loaded Required System Files

The Loader, as the name implies, should load all necessary system files to memory. This is done by parsing the parent file system, reading in each file, placing the contents of the file at a specified location in memory. Each file on the file system has a specific formatted header to describe the type of file being loaded as well as where to place it within memory.

See Section 6 of this specification for the format of this header as well as the file it defines.

The list of file names is currently hard coded within the loader file. The location of this list and the format of this list is not part of this specification.

Each filename must use the same requirements as shown in Section 3.1 for the loader file using the same folder search order as well as case insensitive matching.

The Loader may use Folder/Directory hierarchies if desired. If doing so, the search order as shown in Section 3.1 is only used if the current filename and path given doesn't start at the root of the file system. If the filename uses a direct path from the root of the file system, there is no need to search the folders listed in Section 3.1.

4.8. Interrupt 1Eh Data

Retrieve the Floppy Diskette Information Block from the BIOS as shown in Section 4.3.9 and 4.3.10. This information must be zero when a UEFI BIOS type is found.

4.9. Reset Disk Services

If a Legacy BIOS service was used, you need to reset the services by calling the Legacy BIOS Disk Reset Service.

4.10. BIOS Equipment List

Retrieve the BIOS Equipment List as shown in Section 4.3.14 and 4.3.15. This information must be zero when a UEFI BIOS type is found.

4.11. Disk Drive Parameters

Retrieve the information from each disk drive returned from the BIOS as shown in Section 4.3.22.

4.12. BIOS Time of Day

Retrieve the Date and Time of Day from the BIOS and store as shown in Section 4.3.11. Please note that this should be done as near the end of the Loader process as possible as well as the Kernel should extract this data and update its time keeping mechanism as soon as possible to keep from losing time.

4.13. Detect an APM BIOS

Retrieve the BIOS Advanced Power Management information as shown in Section 4.3.12. This information must be zero when a UEFI BIOS type is found.

4.14. Retrieve Supported Video Modes

Retrieve the current supported graphical video modes as shown in Section 4.3.18 and 4.3.20. After storing this information, the Loader should choose a suitable mode and store that index in the field as shown in Section 4.3.19. The Loader must set the current display to that mode.

It is suggested that a mode change from a text mode to a graphics mode be done near the end of the loader process, so than an error message written to the text based mode won't be missed due to the mode change.

Since the UEFI BIOS no longer supports a text based video mode, the Kernel must not use a text based video mode to display information. Therefore, the kernel must use and will assume a graphic video mode has been set by the loader.

During the loading process, and up to this point, the Loader may watch for a certain key press combination and if found, display the found video modes to the screen, allowing the user to select one of the modes. See Section 5 on how to watch for and the allowed key press combinations.

It is not part of this specification on how to display and allow selection of a found mode.

4.15. Move Loader Transfer Block

Once all information has been retrieved from the BIOS, as well as the Kernel's address has been established, you now must move the temporarily stored Loader Transfer Block to just before the Kernel. This address is specified as KERNEL_ADDRESS + 0x400 - Size_of_Transfer_Block.

4.16. Create Kernel's Address Space

Using the mechanisms of the current processor, you must create enough GDT entries to allow the kernel to have a flat address space as well as the correct CS selector value and EIP register so that when you jump to the kernel, it has full control of the whole address space. The kernel, once loaded, may change the address space as it wishes. However, it is up to the loader to create all segment selector registers to start at address 0x00000000 and use all available physical and non-physical memory.

The Loader should also make sure that the processor's CR4.TSD bit and CR4.VME bit each are clear.

4.17. Create Kernel's Stack

The Loader must set up a stack of four megabytes at physical address STACK_BASE, setting the SS and ESP register pair to use this down-growing 32-bit stack. The Kernel may change this stack location and size.

4.18. Jump to the Kernel

If using the UEFI BIOS, you should once more retrieve the Memory Information as described in Section 4.8 incase you have freed any memory. Please note that you must now use the new address of the Loader Transfer Block since you have now moved it to the new location.

Last thing to do for the Loader is to jump to physical address KERNEL_ADDRESS + 0x400, the address retrieved from the kernel system file as shown in Section 6 of this specification.

5. Key Press Combinations

One of the first things the Loader should do is clear the BIOS' keyboard buffer, so that there are no ghost key presses found.

The Loader then may watch for certain key press combinations and set internal flags when a combination is found. Then when a certain phase of the loader is executed, it may check to see if this internal flag is set and do accordingly.

For example, the user may wish to pause execution at time of video detection, as shown in Section 4.14, to choose a video mode. The user can press a key combination beforehand to have the execution stop at this time. When the key press combination is detected, via a hardware interrupt, the loader will set the internal flag for that combination and continue execution. When the video detection code gets executed, it can check this flag.

The following key press combinations are required, not interrupting the Loader process when pressed, and must set its respective internal flag. However, please note that only the processes marked in the table below as interruptible (X) are required to stop execution and act upon its respective set flag.

Table 5.1: Valid Key Press Combinations

Combination	X	Section	Description
F8	X	4.14	Stop and allow video mode selection
F9		4.xx	Text Mode only (Legacy BIOS Only)
ESC	X		Pause Execution until pressed again

All other key press combinations are to be ignored.

6. Loader System Files

Each system file, including the kernel file, contains a preset header, used to extract information such as where to place the file in memory.

6.1 File Header

The format of this header is shown below.

```
id          dword    ; Signature
location    dword    ; 32-bit physical location to load file
flags       dword    ; file flags
file_crc    dword    ; file's CRC check
comp_type   byte     ; compression type
hdr_crc     byte     ; byte zero check sum of header
file_size   dword    ; size of uncompressed file
reserved    10 dup   ; reserved for future use
```

6.1.1 Signature
The header must start with a 32-bit signature value of 0x46595332 to indicate that it is actually a valid file header.

6.1.2 Location
The Location field is the 32-bit physical memory location to store the uncompressed file data. If this value is 0xFFFFFFFF, any location can be used. However, when using this value, it is up to the loader to maintain a memory map so that files are not overwritten by another file.

6.1.3 Flags
The Flags field holds certain flags related to the loading of this file.

Table 6.1.3: File Flags

Bits	Description
0	Halt on error
1	File is the kernel file
31:2	Reserved

If bit zero is set and there is an error loading this file, the loader is to stop execution, give the error, and halt. Bit 1 indicates that this is the kernel file. The loader is to stop execution and give an error if a file is not found with this bit set, or if more than one file is found with this bit set.

6.1.4 File CRC
This is the 32-bit CRC value calculated from the uncompressed data of the file. The algorithm used is from the standard CRC-32 shown at https://en.wikipedia.org/wiki/Cyclic_redundancy_check and uses a Polynomial of 0x04C11DB7.

6.1.5 Compression Type
This field indicates what kind of compression is used to compress the file's data. The table below shows the possible values.

Table 6.1.5: Compression Type

Value	Description
0	No compression
1	Bz2
any other	Not used and is in error if found

A value of zero indicates no compression was used. The body of the file is a byte-for-byte representation of the data to store to the memory location.

The Bz2 compression is shown at https://en.wikipedia.org/wiki/Bzip2.

6.1.6 Header CRC
This field is a simple zero-sum check of the bytes of this header. If all bytes of the file's header sum to zero, ignoring rollover and carry, then the header is a valid header.

6.1.7 File Size
This is a 32-bit value holding the size of the uncompressed data of this file.

6.1.8 Reserved
This field is used to reserve space for future use. When a file is created, this field should be all zeros.

6.2 File Contents
The file's contents follow the file's header. The contents may be compressed or otherwise modified. See section 6.1.5.

6.2.1 Kernel File
Since the Kernel file may be a PE formatted object file, the first 0x400 bytes are ignored, though still loaded to memory. Therefore, the Loader must jump to KERNEL_ADDRESS + 0x400 when transferring execution to the Kernel. This address is used for all Kernel types no matter the object format used. The Kernel's execution must start at KERNEL_ADDRESS + 0x400.

7. Kernel
*** to be added ***