JPC-RR: User's manual

31. joulukuuta 2014

1 Licence

JPC-RR is licenced under GNU GPL v2. See file "LICENSE"

2 Getting started

2.1 Prerequisites

To get started, you need BIOS image, VGABIOS image and DOS boot floppy and JDK for Java 6 standard edition (later versions should they appear should also work). Note: JRE is not enough.

Note that to play back recorded movies, you need exact same version of BIOS image, VGABIOS image and DOS boot floppy as was used when making the movie (in addition to exact same versions of other needed media).

2.2 Compiling

See compile.sh or compile.bat. The streamtools stuff is only needed for dumping videos.

2.3 Getting started

First you need to get and make some important images. Obtain BIOS image, VGABIOS image and DOS boot floppy from somewhere. After starting the emulator, use Drives -> Import Image to import the images (ignore the error about no BIOS images being found).

2.4 Running emulator

There is premade autoexec script called assemble.jpcrrinit that has fairly reasonable defaults. To use it:

```
java JPCApplication -autoexec assemble.jpcrrinit
```

The script pops up settings for new emulated PC (if you want to load savestate, click cancel). Select BIOS and VGABIOS for BIOS and VGABIOS image (they should be already selected), DOSfloppy for fda (boot device should be set to fda) and game image as some HD drive

2.5 Bootup tips

- Putting the game as hdd (the fourth hard disk slot) causes boot to be bit faster.
- Some BIOS versions have "press F12 to select boot device". Hit <enter> from emulated keyboard and that prompt will go away in about half emulated second (it stays several emulated seconds otherwise).
- If game doesn't need lots of memory, hitting F5 to skip intialization files is fastest. If it does need more memory, run config.sys commands but not autoexec.bat.
- Some DOS disks have DOSIDLE with them, don't use it as it messes badly with emulator.

3 Making JPC-RR format images from raw images

Due to various factors, JPC-RR can't use raw image files directly but requires its own image format.

3.1 Importing images from GUI:

Use Drives -> Import Image to import existing directories or image files. Dialog prompting parameters will be displayed. When importing floppy images, check "standard geometry" if possible, that enables geometry autodetection, which is reasonable virtually all of the time it is offered.

3.2 Notes

- If making image from directory, the names of the files must conform to FAT naming restrictions (8+3 character names, no spaces, etc). Avoid filenames with non-ASCII characters.
- The DOS limit of 112 or 224 files for floppies does not apply to images created from directory trees. The minimum limit value used is 512. If even that isn't enough, the limit is automatically increased to fit all the needed directory entries.
- Making boot disks from tree does NOT work. Even if you got system boot files there, it still won't work.
- Only floppy disks and hard drives can be made from directory trees. BIOS images and CDROM images require image file.
- Avoid floppies with custom geometry (floppy geometry does affect disk ID). Disks with over 63 sectors per track don't work with DOS. Wheither disks with over 127 tracks per side work with DOS is unknown. Also avoid 1024-tracks per side HDDs.
- The geometry limits are: 2-1024 tracks per side for HDD, 1-256 tracks per side for floppy. 1-63 sectors per track for HDD, 1-255 sectors per track for floppy. 1-16 sides for HDD, 1 or 2 sides for floppy. This gives size limit of 65280KiB for floppy disks (but note the DOS limit!) and 516096KiB for HDDs.
- There are multiple image file contents that represent the same image. The one with smallest size is picked when creating image.
- Note: Although the IDs are 128 bits long, they are not MD5 hashes.

3.3 Importing from command line

There is tool called ImageMaker that can make JPC-RR images from raw images. Each image has format, ID an name. Format and name are specified when making image. ID is automatically calculated from format and contents. Name does not affect the ID but is purely for convience so one doesn't have to specify long image IDs manually.

3.3.1 Syntax

The syntax for ImageMaker when making images is:

```
$ java ImageMaker <format> [<options>...] <destination> <source> <name>
```

<destination> is file name for JPC-RR format image to write. <source> is either name of regular file (raw image file) or name of directory tree with files (supported for making floppy or hard disk images only). In case of directory tree, the files are layout deterministically to disk, so the ID will always be the same for given geometry and type. <name> is name to give to disk. <format> is one of:

```
-BIOS BIOS image (note: VGABIOS is also of this type).
-CDROM CD-ROM image.
-HDD=cylinders,sectors,heads Hard disk with specified geometry.
-floppy=tracks,sectors,sides Floppy disk with specified geometry.
-floppy160 160KiB floppy (40 tracks, 8 sectors, Single sided).
-floppy180 180KiB floppy (40 tracks, 9 sectors, Single sided).
-floppy320 320KiB floppy (40 tracks, 8 sectors, Double sided).
```

```
-floppy360 360KiB floppy (40 tracks, 9 sectors, Double sided).
-floppy410 410KiB floppy (41 tracks, 10 sectors, Double sided).
-floppy420 420KiB floppy (42 tracks, 10 sectors, Double sided).
-floppy720 720KiB floppy (80 tracks, 9 sectors, Double sided).
-floppy800 800KiB floppy (80 tracks, 10 sectors, Double sided).
-floppy820 820KiB floppy (82 tracks, 10 sectors, Double sided).
-floppy830 830KiB floppy (83 tracks, 10 sectors, Double sided).
-floppy880 880KiB floppy (80 tracks, 11 sectors, Double sided).
-floppy1040 1040KiB floppy (80 tracks, 13 sectors, Double sided).
-floppy1120 1120KiB floppy (80 tracks, 14 sectors, Double sided).
-floppy1200 1200KiB floppy (80 tracks, 15 sectors, Double sided).
-floppy1440 1440KiB floppy (80 tracks, 18 sectors, Double sided).
-floppy1476 1476KiB floppy (82 tracks, 18 sectors, Double sided).
-floppy1494 1494KiB floppy (83 tracks, 18 sectors, Double sided).
-floppy1600 1600KiB floppy (80 tracks, 20 sectors, Double sided).
-floppy1680 1680KiB floppy (80 tracks, 21 sectors, Double sided).
-floppy1722 1722KiB floppy (82 tracks, 21 sectors, Double sided).
-floppy1743 1743KiB floppy (83 tracks, 21 sectors, Double sided).
-floppy1760 1760KiB floppy (80 tracks, 22 sectors, Double sided).
-floppy1840 1840KiB floppy (80 tracks, 23 sectors, Double sided).
-floppy1920 1920KiB floppy (80 tracks, 24 sectors, Double sided).
-floppy2880 2880KiB floppy (80 tracks, 36 sectors, Double sided).
-floppy3120 3120KiB floppy (80 tracks, 39 sectors, Double sided).
-floppy3200 3200KiB floppy (80 tracks, 40 sectors, Double sided).
-floppy3520 3520KiB floppy (80 tracks, 44 sectors, Double sided).
-floppy3840 3840KiB floppy (80 tracks, 48 sectors, Double sided).
```

3.3.2 Other options

```
-volumelabel=label Give specified volume label (affects ID). Only meaningful when making image out of
-timestamp=YYYYMMDDHHMMSS Give specified timestamp for files (affects ID). Only meaningful when making
```

3.3.3 Image information

When invoked as:

```
$ java ImageMaker <imagefile>
```

Variety of information about image is displayed (especially for floppies/HDDs). Two important fields are calculated and claimed disk ID. They should be the same. If they are not, then the image file is corrupt (sadly, imagemaker has bugs and bugs that cause it to write corrupt images have been seen).

3.4 Advanced: The disk ID algorithm

The disk ID is calculated as:

```
Skein-256-128-deprecated(<typecode>|<geometry>|<image>)
```

Where Skein-256-128-deprecated is Skein hash function with 256-bit internal state and 128-bit output using the deprecated rotation constants (as specified in Skein hash function reference documentation versions 1.0 and 1.1). The <image> is the whole image, including parts not stored in image file. The reason for keeping using the deprecated constants are:

- Changing the constants would change the IDs, which would invalidate existing images
- This is not about cryptographic security
- The new constants don't improve security that much anyway.

3.4.1 Floppies and HDDs

Floppies have <typecode> value 0 (single byte) and HDDs have 1 (single byte). <geometry> is as follows (this is exactly the same form as it appears in image header):

```
Byte 0 bits 0-1: Bits 8-9 of track count per side - 1.
Byte 0 bits 2-5: Head count - 1.
Byte 0 bits 6-7: Reserved, must be 0.
Byte 1: Bits 0-7 of track count per side - 1.
Byte 2: Sector count per track - 1.
```

3.4.2 CD-ROM and BIOS images

CD-ROMs have <typecode> value 2 (single byte) and BIOS images have 3 (single byte). <geometry> is blank.

3.5 Advanced: Disk Image format

The disk image consists of following parts, concatenated in this order without padding:

- Magic
- Disk ID
- Type code
- Disk name length
- Disk name
- type-specific geometry/size data
- Actual image data
- Comments

3.5.1 Magic

Magic in disk image files is following 5 bytes: "IMAGE"

3.5.2 Disk ID

Disk ID is given as 16 bytes, encoding the 128-bit disk ID.

3.5.3 Type code

Type code is single byte. 0 for floppies, 1 for HDDs, 2 for CD-ROMs and 3 for BIOS images. Other values are reserved.

3.5.4 Disk name length

Obsolete. Disk name length is given as two-byte big-endian value. New images should have 0 here.

3.5.5 Disk name

Ignored. Name field is there for backward compatiblity. Disk name length gives length of this field in bytes.

3.5.6 Type-specific geometry/size data (floppies and HDDs)

Floppies and HDDs have 3-byte geometry data:

```
Byte 0 bits 0-1: Bits 8-9 of track count per side - 1.
Byte 0 bits 2-5: Head count - 1.
Byte 0 bits 6-7: Reserved, must be 0.
Byte 1: Bits 0-7 of track count per side - 1.
Byte 2: Sector count per track - 1.
```

3.5.7 Type specific-geometry/size data (CD-ROMs)

CD-ROMs have 4-byte big-endian sector (512 bytes!) count.

3.5.8 Type specific-geometry/size data (BIOS images)

BIOS images have 4-byte big-endian byte (not sector or block) count.

3.5.9 Actual image data (floppy/HDD)

Floppy or HDD imagedata consists of following subparts:

- Storage method
- Sectors present
- Image data header
- Image data

Storage method is single byte. Sectors present gives number of last nonzero sector + 1 (zero if image is all zeroes)

3.5.10 Floppy/HDD storage method 0: Raw storage

This storage method has empty header. Image data is raw dump of first sectors present sectors.

3.5.11 Floppy/HDD storage method 1: Sectormap

Image data header contains bitfield with just enough bytes to have one bit per present sector. The order of bits is such that number of bit corresponding to each sector in byte is sector number modulo 8 and byte number is floor of sector number divided by 8 when sector numbers are counted from zero. If bit corresponding to sector is set, then the sector is present in image data, otherwise it is absent and assumed to be all-zeroes.

Image data contains dumps of all present sectors in order of increasing sector number.

3.5.12 Floppy/HDD storage method 2: Extent first sector zero

Image data is empty as storage-specific data is mangled with image data. The image data alternates between blocks encoding zero sectors and blocks encoding nonzero sectors. The first block encodes zero sectors.

Block encoding zero sectors consist of single 1-4 byte little-endian value encoding number of sectors in block - 1. Number of bytes is determined by sectors present value. It is 1 for 1-256 sectors, 2 for 257-65536, 3 for 65537-16777216 and 4 for more than 16777216. All sectors in block are filled with zeroes and are not stored.

Block encoding nonzero sectors has same block count as zero sector block but is followed by the sectors stored raw.

3.5.13 Floppy/HDD storage method 3: Extent first sector nonzero

Same as storage method 2 but first block is nonzero sector block.

3.5.14 Actual image data (CD-ROMs and BIOS images)

These store image data raw. The amount of data is specified by sector/byte count.

3.5.15 Comments

Comments are given as list of strings, with UTF-8 encoded strings following 2-octet big-endian length. Comment list is terminated by entry with length 0 (0x00 0x00). Comments are optional and may be absent.

4 The actual emulator

The actual emulator is invoked as:

```
$ java JPCApplication <options>...
```

The valid options are:

```
-autoexec <script> Execute contents of specified file as commands when starting up.
-noautoexec Don't run autoexec files.
-norenames Copy&Delete files instead of renaming. Mainly meant for debugging copy&delte code.
-imagemaker <options> Run in image maker mode (run with parameter '-imagemaker' with no further parame
```

If no arguments are given, defaults of autoexec file of 'assemble.jpcrrinit' are used.

4.1 Command line

When emulator is started, command line comes up. Following commands are known:

- 'exit': exit immediately. Dumps in progress are gracefully closed.
- 'kill': Save stack traces and kill the emulator (for debugging only). Any dumps in progress are likely corrupted.
- 'library <library>': set library directory to <library>.
- 'load <plugin>': Load plugin (no arguments)
- 'load <plugin>(<arguments>)': load plugin with arguments.
- 'command <command> [<arguments>...]': Invoke command via external command interface.
- 'call<command> [<arguments>...]': Invoke command via external command interface and print return values.
- 'lsdisks [<filename>]' Print listing of all known disks. If <filename> is specified, save output to specified file.
- 'diskinfo [<filename>] <imagename>' Print Information about <imagename> (can be disk name or ID). If <filename> is specified, save output to specified file.

When one gets command line, its useful to load some plugins. See section about plugins. Note: Load runner plugin (PCControl/PCRunner and so) last, as some runners like to start PC immediately.

4.2 PC settings dialog notes

- CPU divider base frequency before division is 1GHz.
- Images can be specified by name or by ID. Name is relative to library directory. If the image is in subdirectory of image directory, the directory separator is is '/' regardless of what the host OS uses.
- CD-ROM and hdc are mutually exclusive
- Modules is comma-seperated list of modules to load. To pass arguments to some modules, enclose the arguments in (). Same module can be specified twice only if parameters differ.
- Setting boot device doesn't work with some BIOS versions. Those versions prompt the boot device anyway.

4.3 Audio output channels

PC can have one or more audio output channels. The name of audio output associated with PC speaker is: 'org.jpc.emulator.peripheral.PCSpeaker-0'. Modules that have audio outputs get channel names of form <classname>-<sequential>, where <classname> is name of main module class and sequential is number starting from zero. Note that same module can have multiple output channels. If multiple modules of same class request audio outputs, the <sequential> values of subsequent module start where previous left off.

4.4 Plugins

Plugins actually execute the tasks of the emulator. They can be loaded using "load <plugin>" or 'load <plugin>(<arguments>)" from command line.

Different Plugins using the same output (like running PCMonitor and RAWVideoDumper) should not conflict because connector output hold locking is desinged to handle multiple readers.

If no plugin used requires GUI, then the emulator can be run without having GUI available.

4.4.1 plugin: org.jpc.plugins.PCControl

Takes optionally 'extramenu=<file>' and 'uncompressedsave=1', requires and uses GUI.

Runs the PC emulator core. Has capability to start/stop emulation, breakpoint after certain time or start/end of VGA vertical retrace. Also can create, savestate and loadstate PC emulation. Memory dumping is supported.

'extramenu=<file>' causes Plugin to load extra menu entries from <file>. 'uncompressedsave=1' causes savestates to be written uncompressed (useful if they are stored in VCS supporting delta compression).

4.4.2 plugin: org.jpc.plugins.PCRunner

Takes 'movie=<file>' as argument and optionally 'stoptime=<time>' Does not require nor use GUI.

Loads PC from savestate and just runs it. CTRL+C to quit. Also automatically quits once stoptime is reached.

4.4.3 plugin: org.jpc.plugins.PCMonitor

No arguments, requires and uses GUI.

VGA monitor for emulated PC.

4.4.4 plugin: org.jpc.plugins.VirtualKeyboard

No arguments, requires and uses GUI.

On-screen keyboard for emulated PC.

4.4.5 plugin: org.jpc.plugins.PCStartStopTest

No arguments, requires and uses GUI.

Small plugin testing remote PC start/stop. Also supports sending some common keypresses.

4.4.6 plugin: org.jpc.plugins.RAWVideoDumper

Takes 'rawoutput=<file>' as argument. Does not require nor use GUI.

Dumps all generated frames to RAW file <file>. Rawoutput is required. The raw file consists of concatenation of zlib streams. The uncompressed stream is concatenation of time skips (FFh FFh FFh FFh), each acting as time offset of 2^32-1 nanoseconds and saved frames. The saved frame has time offset in nanoseconds (big endian) as first four bytes (must be at most 2^32-2, as 2^32-1 is reserved for time skip). The next two bytes are big-endian width, next two big-endian height. Finally frame has 4 * width * height bytes of data that encodes pixels using 4 bytes per pixel, in left-to-right, up-to-down order. Byte 0 of each pixel is reserved, byte 1 is the red channel, byte 2 is green channel and byte 3 is blue channel.

Dumping to pipe is supported.

4.4.7 plugin: org.jpc.plugins.RAWAudioDumper

Takes 'src=<name of audio output channel>', 'file=<output-filename>' and 'offset=<offset>' as arguments, separated by ','. Does not require nor use GUI.

Dumps output from specified audio output channel (src, mandatory) to RAW-format file (file, mandatory). The resulting file consists of records, 4 or 8 bytes each. 4 byte record consists of 0xFF 0xFF 0xFF 0xFF and means to increase next time delta by $2^{32} - 1$ns. Otherwise record is 8 bytes. Each 8 byte record has three fields. First 4 byte unsinged big endian timedelta value (in nanoseconds, must be smaller than $2^{32} - 1$), then 2 byte signed big endian new left channel volume, then 2 byte signed big endian new right channel volume. Optionally 'offset' can be set to positive value (in nanoseconds) to delay the audio by.

4.4.8 plugin: org.jpc.plugins.LuaPlugin

Takes 'kernel=<name of lua kernel file>', other parameters are passed to kernel, requires and uses GUI. Lua VM for executing scripts.

4.4.9 plugin: org.jpc.plugins.JoystickInput

No parameters. Displays window for sending joystick input.

5 Modules

5.1 org.jpc.modules.Joystick:

- Arguments: none.
- Resources: I/O port 0x201

Emulates joystick game port.

5.2 org.jpc.modules.SoundCard

- Arguments: Optional resources specification
- Resources (defaults): I/O port 0x220-0x22F, IRQ 5, DMA 1, DMA 5

Emulates Sound card.

5.3 org.jpc.modules.GMIDIInterface

- Arguments: Optional resources specification
- Resources (defaults): I/O port 0x330-0x331, IRQ 9

Emulates General MIDI interface.

5.4 org.jpc.modules.FMCard

- Arguments: Optional resources specification
- Resources (defaults): I/O port 0x338-0x33B

Emulates FM card.

5.5 org.jpc.modules.BasicFPU:

- Arguments: none.
- Resources: None.

Crude FPU (x87) emulator.

6 Hacks

Hacks are saved to savestates but not movies.

6.1 NO_FPU

Force bit 1 of physical address 0x0410 to zero, signaling that the system has no FPU. BIOS assumes system has FPU but some games use that bit to detect FPU, trying to use it if it is "present". Try this if game startup hangs with lots of trying to use FPU but not present errors. Don't use if there is FPU present. Needed to get games like Blake Stone / Wolfenstein 3-D to work (FPU emulator allows it to start but causes graphical glitches).

6.2 VGA_DRAW

Update basic VGA parameters before vretrace, not after it. Some games (e.g. Commander Keen 4) don't like if this isn't done and some games (e.g. Mario & Luigi) don't like if it is done. Wrong value manifests as jerky scrolling (scrolling back and forth and fixed statusbars move).

7 Some error messages and explanations

- <filename> is Not a valid image file
- <filename> is not image file
- <filename> claims to be floppy with illegal geometry: <x> tracks, <y> sides and <z> sectors.
- <filename> claims to be HDD with illegal geometry: <x> tracks, <y> sides and <z> sectors.
- Can't read disk image sector map.
- Can't read disk image extent.

Code expects <filename> to be valid JPC-RR format image, but it isn't JPC-RR image at all or its corrupt.

- <filename> is image of unknown type.
- <filename> has unrecognized geometry <x> <y> <z>

Possibly corrupt image, not JPC-RR image, or JPC-RR image from future version containing something current version can't comprehend.

- Invalid format specifier <something>.
- Invalid syntax of –floppy= or –HDD= option.
- Invalid format specifier/option <something>.

Invalid option or format specifier was given. Check for typos.

- java ImageMaker [<options>...] <format> <destination> <source> <diskname>

Check syntax of command. Especially that diskname is present!

- The image has <nnn> sectors while it should have <yyy> according to selected geometry.
- Raw image file length not divisible by 512.
- Trying to read sector out of range.

The selected geometry is wrong or raw image is incomplete.

- Invalid disk name (Should not happen!).
- Invalid geometry to be written.

This is a very likely a bug in program.

- What the heck <filename> is? It's not regular file nor directory.

That sort of file can't be used as input for image making, or the file just doesn't exist.

- BIOS images can only be made out of regular files.
- CD images can only be made out of regular files.

Source image specified is not regular file, but image of that type can't be made of anything else.

- Can't read raw bios image file.

- Can't read sector <nnn> from image.

Reading the raw image file failed for some reason.

- Bad library line: "<something>". Ignored.

Syntax error in image library.

- Removing image <something> a.k.a. "<something>" as it no longer exists.

The image file no longer exists so it gets removed from library.

- Removing image <something> a.k.a. "<something>" due to <some> conflict.

Image library code killed some image from library due to some kind of conflict with image being added.

- Too much data to fit into given space.

The tree you gave contains takes just too much space to fit into disk of this size.

8 Advanced: Savestate/movie format

8.1 Special character classes

8.1.1 SPACE

Following Unicode codepoints (encoded as UTF-8) are interpretted as space characters:

- Codepoints 0x20, and 0x09.
- Codepoints 0x1680, 0x180E, 0x2028, 0x205F and 0x3000
- Codepoints 0x2000-0x200A.

8.1.2 LINEFEED

Following byte sequences are interpretted as linefeeds (line change):

- Byte 0x0A (UTF-8 encoded codepoint 0x0A)
- Byte 0x0D (UTF-8 encoded codepoint 0x0D)
- Byte 0x1C (UTF-8 encoded codepoint 0x1C)
- Byte 0x1D (UTF-8 encoded codepoint 0x1D)
- Byte 0x1E (UTF-8 encoded codepoint 0x1E)
- Bytes 0xC2 0x85 (UTF-8 for unicode control character NL, codepoint 0x85)
- Bytes 0xE2 0x80 0xA9 (UTF-8 encoded codepoint 0x2029)

8.2 JRSR archive

JRSR archive format packs multiple text archive members to text archive. It does not support binary members. JRSR archives have first five or six bytes form the magic. It is "JRSR" followed by LINEFEED character There are four kinds of lines after that (lines are terminated by LINEFEED byte/bytes):

- Start member
- Member line
- End member
- Blank line

Sequencing rules are as follows: Start member is allowed anywhere (after magic). Member line is allowed only inside member (member started but not ended). End member is only allowed inside member. End of file is only allowed outside member. Blank line is allowed anywhere after magic.

8.2.1 Start member

Start member line is given as "!BEGIN" <SPACE>+ <membername> <LINEFEED>. <SPACE>+ any number of SPACE characters at least one and <LINEFEED> is LINEFEED chacter. The member name is UTF-8 encoded and maximum allowed line length is 2048 bytes (including LINEFEED, which means name is limited to 509-2040 codepoints depending on characters used). Starting member inside another implicitly ends the previous member.

8.2.2 Member line:

Member line is given as "+"<content><LINEFEED>. It gives another line for member contents. <content> is passed raw to layers above (followed by line termination)

8.2.3 End member

End member line is given as "!END"<LINEFEED>. It ends the current member. The following line can only be start member line or file may end.

8.2.4 Blank line

Blank line is given as <LINEFEED>. Lines like that are ignored.

8.3 Four-to-Five encoding

Binary members are encoded into text by so-called four-to-five encoding. This encoding can encode single byte to two, two bytes to three, three bytes to four and four bytes to five. Four-to-five encoding has five kinds of blocks. All SPACE and LINEFEED characters are completely ignored, even in middle of blocks.

8.3.1 End stream block

End stream block is encoded as '!'. It ends the stream instantly. There is also implicit end of stream at end of input to decoding.

8.3.2 Other four block types

Other four block types take the value to be encoded, read it as big-endian value. Then they write it as base-93 big-endian value. Then length specific constants are added to digits of that number to yield ASCII values for characters (those are stored in order):

To encode	1st char.	2nd char.	3rd char.	4th char.	5th char.
1 byte	34	34	-	-	-
2 bytes	37	34	34	-	-
3 bytes	45	34	34	34	-
4 bytes	66	34	34	34	34

Blocks which encode values greater than what is possible for value of that length are fatal errors.

8.4 Line component encoing

Line component encoding sits on top of UTF-8 encoding. Line component encoding encodes non-empty 1-D array of non-empty strings into line, and thus array of those into member. Empty lines or lines that don't contain any components are ignored. Line starts with depth value of 0 and must end with depth value of zero.

Components are seperated by component separators. Empty components are ignored. Following codepoints are separators on depth 0 if not escaped:

- Codepoint of '('. The depth is read pre-increment.
- Codepoint of ')'. The depth is read post-decrement.
- Any SPACE character

The following characters are special:

- '('. Increments depth by 1 if not escaped (and appears in component).
- ')'. Decrements depth by 1 if not escaped (and appears in component). Depth going negative is an error.
- '\'. Next character is interpretted as literal. Error if at end of line.

Otherwise, characters are interpretted as literals and appear in components. Depth must be zero at end of line.

8.5 Header section:

Header section is in archive member "header". It uses line component encoding. The first component of each line is name of header, and subsequent ones are arguments. How many parameters are expected is dependent on what header it is:

8.5.1 PROJECTID header:

- Header name: "PROJECTID"
- Components: 2
- Argument #1: <project-id-string>
- Mandatory: Yes

Gives project ID. Project ID is generated when PC is assembled and is then preserved in save states. It is used for computing rerecord counts. Emulator treats it as opaque string, the IDs it generates are formed by 48 random hexadecimal digits.

8.5.2 SAVESTATEID header:

- Header name: "SAVESTATEID"
- Components: 2
- Argument #1: <savestate-id-string>
- Mandatory: No

Gives save state ID. Each save state has its own save state ID. Treated as opaque string, but generated as 48 random hexadecimal digits. The presence of this header signals whether there is save state to be loaded. If this header is present, save state load will be attempted. If absent, save state is not to be loaded even if present (and correct savestate load would be technically impossible anyway).

The value is used to prevent loading incompatible save states in preserve event stream mode and also to find the point in event stream where one left off.

8.5.3 RERECORDS header:

- Header name: "RERECORDS"
- Components: 2
- Argument #1: <rerecords>
- Mandatory: Yes

Gives rerecord count. PC assembly (except when loading save state) initializes current rerecord count to zero. Must be non-negative and decimal number using ASCII digit characters.

```
On loading save state:
1) If project ID matches with previous:
1a) If loaded rerecord count is larger or equal to current rerecord count:
1a-a) Current rerecord count is loaded rerecord count + 1.
1b) Otherwise
1b-a) Current rerecord count increments by 1.
2) Otherwise
2a) Current rerecord count is loaded rerecord count + 1.
```

The current rerecord count at time of save is saved to save state.

8.5.4 SYSTEM header:

- Header name: "SYSTEM"
- Components: 2
- Argument #1: <system-id-string>
- Mandatory: No

Gives system this movie/save is for. The only currently recognized values are "PC-JPC-RR-r10" and "PC-JPC-RR-r11.3" (if this header is absent, use default system). Invalid values trigger error on load time. The "PC-JPC-RR-r11.3" should be used if INITIALSTATE headers are present to avoid earlier versions from getting confused by those.

8.5.5 AUTHORS header:

- Header name: "AUTHORS"
- Components: 2 or more
- Arguments: free form
- Mandatory: No

Gives authors of run. Each argument gives one author (who has full name but no nickname). May be present multiple times.

8.5.6 AUTHORNICKS header:

- Header name: "AUTHORNICKS"
- Components: 2 or more
- Arguments: free form
- Mandatory: No

Gives authors of run. Each argument gives one author (who has nickname but no full name). May be present multiple times.

8.5.7 AUTHORFULL header:

- Header name: "AUTHORFULL"
- Components: 3
- Arguments: free form
- Mandatory: No

Gives author of run. First argument is full name of author, and second is nickname of author. May be present multiple times.

8.5.8 COMMENT header:

- Header name: "COMMENT"
- Components: 2 or more
- Arguments: free form
- Mandatory: No

Various kinds of free form data. Not parsed further by emulator.

8.5.9 INITIALSTATE header:

- Header name: "INITIALSTATE"
- Components: 2 or more
- Arguments: Name of initial state
- Mandatory: No

Denotes that “initialization-”<name> is valid initial state.

8.6 Initialization segment:

If SAVESTATEID header isn't present (not a save state), member "initialization" (or “initialization-”<name>) gives PC initialization parameters for assembling the PC. It is present anyway even if SAVESTATEID is present (savestate).

Following parameters are used (space separates components):

```
"BIOS" <id>
```

Gives Image ID of main system BIOS (mandatory)

```
"VGABIOS" <id>
```

Gives Image ID of VGA BIOS (mandatory).

```
"HDA" <id>
```

Gives Image ID of hda. Present only if system has hard disk hda.

```
"HDB" <id>
```

Gives Image ID of hdb. Present only if system has hard disk hdb.

```
"HDC" <id>
```

Gives Image ID of hdc. Present only if system has hard disk hdc.

```
"HDD" <id>
```

Gives Image ID of hdd. Present only if system has hard disk hdd.

```
"DISK" <num> <id>
```

Gives Image ID of disk in slot <num>. Slot number must be non-negative.

```
“DISKNAME” <num> <name>
```

kGives image name of disk in slot <num>. Slot number must be non-negative. The slot must be previously declared using “DISK”.

```
"FDA" <num>
```

Gives Image slot to initially put into floppy drive fda. Disk must be of floppy type. If none present, no disk is initially put there.

```
"FDB" <num>
```

Gives Image slot to initially put into floppy drive fdb. Disk must be of floppy type. If none present, no disk is initially put there.

```
"CDROM" <num>
```

Gives Image slot to initially put into CD-ROM drive hdc. Not allowed if hard disk hdc is present. Disk must be of CD-ROM type. If none present no disk is initially put there.

`"INITIALTIME" <time>`

Number of milliseconds since Unix epoch to system start up time. Allowed range:
0-4102444799999. Mandatory.

`"CPUDIVIDER" <divider>`

Set CPU frequency divider (dividing the 1GHz master clock). Allowed range is 1-256. Mandatory.

`"MEMORYSIZE" <pages>`

Number of 4KiB pages of RAM memory. Allowed range 256-262144. Mandatory.

`"BOOT" <device>`

Set boot device. Valid devices are "FLOPPY" (boot from fda), "HDD" (boot from hda) and "CDROM" (boot from CD).

`"LOADMODULEA" <module> <parameters>`

Load module <module> with parameters <parameters>.

`"LOADMODULE" <module>`

Load module <module> with no parameters

`“FPU” <fpu>`

Use class <fpu> as FPU emulator.

`“IOPORTDELAY”`

Use I/O port delay emulation (each I/O port read/write takes 666ns).

`“VGAHRETRACE”`

Emulate VGA horizontal retrace.

8.7 Event record format:

Event record is in archive member "events". It uses line component encoding. Each line gives an event. First component of each line gives time stamp. These timestamps MUST be in increasing order and all MUST be non-negative. Time stamp time unit is exactly 1 nanosecond of emulated time.

The second component of each line is name of class to dispatch to. Further components are passed as-is to event handlers. Classes with names consisting only of uppercase A-Z and 0-9 are special and reserved. It is error to encounter unknown such special class.

8.7.1 Savestate event

- Dispatch to: SAVESTATE
- Argument #1: Savestate id
- Argument #2 (optional): Rerecord count at time of saving savestate

Signals that savestate has occured here. The save state IDs MUST be unique in entire event stream. The second argument to savestate (if present) is rerecord count at time of saving that savestate (useful for calulating rerecord count of movie starting from savestate). No time restrictions

8.7.2 Option event

- Dispatch to: OPTION
- Argument #1: “ABSOLUTE” or “RELATIVE”

Controls various options. “ABSOLUTE” turns on absolute mode (default) where event timestamps are absolute. “RELATIVE” turns on relative mode where event timestamps are relative to last event in stream. The OPTION event itself is not affected by timing change. No time restrictions. Unknown arguments are errors.

8.7.3 Keyboard keypress/keyrelease event:

- Dispatch to: org.jpc.emulator.peripheral.Keyboard
- Argument #1: Fixed: "KEYEDGE"
- Argument #2: Key number. Valid values are 1-83, 85-95, 129-197 and 199-223

Send key press or key release. Keys work in toggle button manner. The event time must be multiple of 66 666, and must not be less than 60 * 66 666 TUs after last PAUSE event, 20 * 66 666 TUs after last KEYEDGE on key >128 and 10 * 66 666 TUs after last KEYEDGE on key <128.

8.7.4 Pause event:

- Dispatch to: org.jpc.emulator.peripheral.Keyboard
- Argument #1: Fixed: "PAUSE"

Send pause key event. The time restrictions are identical to KEYEDGE event.

8.7.5 Mouse button event:

- Dispatch to: org.jpc.emulator.peripheral.Keyboard
- Argument #1: Fixed: "MOUSEBUTTON"
- Argument #2: Number of button to release or press (0-4)

Presses or releases the designated mouse button.

8.7.6 X mouse motion event:

- Dispatch to: org.jpc.emulator.peripheral.Keyboard
- Argument #1: Fixed: "XMOUSEMOTION"
- Argument #2: Number of units to move (-255 - 255)

Move the mouse in X direction by specified amount. Positive is right.

8.7.7 Y mouse motion event:

- Dispatch to: org.jpc.emulator.peripheral.Keyboard
- Argument #1: Fixed: "YMOUSEMOTION"
- Argument #2: Number of units to move (-255 - 255)

Move the mouse in Y direction by specified amount. Positive is up.

8.7.8 Z mouse motion event:

- Dispatch to: org.jpc.emulator.peripheral.Keyboard
- Argument #1: Fixed: "ZMOUSEMOTION"
- Argument #2: Number of units to move (-7 - 7)

Move the mouse in Z direction (scrollwheel) by specified amount.

8.7.9 Joystick button event:

- Dispatch to: org.jpc.modules.Joystick
- Argument #1: “BUTTONA”, “BUTTONB”, “BUTTONC” or “BUTTOND”
- Argument #2: “0” if released, “1” if pressed

Send button down/up event. No time restrictions.

8.7.10 Joystick axis event:

- Dispatch to: org.jpc.modules.Joystick
- Argument #1: “AXISA”, “AXISB”, “AXISC” or “AXISD”
- Argument #2: Multivibrator unstable state length in ns.

Set amount of time multivibrator remains in unstable state. No time restrictions.

8.7.11 Reboot:

- Dispatch to: org.jpc.emulator.PC$ResetButton
- No arguments

Reboots the PC.

8.7.12 Fda disk change:

- Dispatch to: org.jpc.emulator.PC$DiskChanger
- Argument #1: Fixed: "FDA"
- Argument #2: Number of image slot to put there.

The disk number MUST be -1 or valid disk number. -1 MUST NOT be used if there is no disk in floppy drive A. This event causes specified disk to be placed to FDA or FDA disk to be ejected with no replacement if disk number is -1. The specified disk if not -1 must be of floppy type. The specified disk if valid must not be in any other drive.

8.7.13 Fdb disk change:

- Dispatch to: org.jpc.emulator.PC$DiskChanger
- Argument #1: Fixed: "FDB"
- Argument #2: Number of image slot to put there.

The disk number MUST be -1 or valid disk number. -1 MUST NOT be used if there is no disk in floppy drive B. This event causes specified disk to be placed to FDB or FDB disk to be ejected with no replacement if disk number is -1. The specified disk if not -1 must be of floppy type. The specified disk if valid must not be in any other drive.

8.7.14 Change CDROM:

- Dispatch to: org.jpc.emulator.PC$DiskChanger
- Argument #1: Fixed: "CDROM"
- Argument #2: Number of image slot to put there.

The disk number MUST be -1 or valid disk number. -1 MUST NOT be used if there is no disk in CD-ROM. This event causes specified disk to be placed to CD-ROM or CD-ROM disk to be ejected with no replacement if disk number is -1. The specified disk if not -1 must be of CD-ROM type.

This event has no effect if CD-ROM is locked.

8.7.15 Write protect floppy:

- Dispatch to: org.jpc.emulator.PC$DiskChanger
- Argument #1: Fixed: "WRITEPROTECT"
- Argument #2: Number of image slot to manipulate

Write protects specified disk. The disk MUST NOT be in any drive and MUST be valid floppy-type disk.

8.7.16 Write unprotect floppy:

- Dispatch to: org.jpc.emulator.PC$DiskChanger
- Argument #1: Fixed: "WRITEUNPROTECT"
- Argument #2: Number of image slot to manipulate

Disables write protection specified disk. The disk MUST NOT be in any drive and MUST be valid floppy-type disk.

8.8 Diskinfo sections

Diskinfo sections are named “diskinfo-”<id of disk>. They use line component encoding, fieldtype being first component on each line (value being the second). Following fields are defined:

8.8.1 TYPE

Gives type of image. Possible values are

- “FLOPPY” (floppy disk)
- “HDD” (Hard disk)
- “CDROM” (CD-ROM)
- “BIOS” (BIOS/VGABIOS image)
- “UNKNOWN” (what the heck is this???)

8.8.2 ID

Gives ID of disk.

8.8.3 IMAGELENGTH

(BIOS images only) Gives length of BIOS image

8.8.4 IMAGEMD5

MD5 of raw disk/BIOS image without any headers or trailers.

8.8.5 TOTALSECTORS

(FLOPPY/HDD/CDROM images only) Number of total sectors on disk.

8.8.6 TRACKS

(FLOPPY/HDD images only) Number of tracks on disk per side (1-256 for floppy, 1-1024 for HDD).

8.8.7 SIDES

(FLOPPY/HDD images only) Number of sides on disk (1 or 2 for floppy, 1-16 for HDD).

8.8.8 SECTORS

(FLOPPY/HDD images only) Number of sectors per track (1-255 for floppy, 1-63 for HDD).

8.8.9 COMMENT

Line from image comment block. Usually give data about files image has. May or may not be present (multiple times)

8.9 Output info

Output info is stored in section "output-info". Its relatively new, so it might not be present (then the contents have to be guessed based on modules present). Each line gives information about one output, first field being the type of output.

8.9.1 Video output

For video output, there are no parameters so line is just "VIDEO" (one component).

8.9.2 Audio output

For audio output, the only parameter is name of output, so first component is "AUDIO" and second component is name of audio output.

8.10 Savestates

Actual savestate format is not documented here. It is close to impossible to comprehend without access to emulator source anyway.

9 Advanced: Making class dumpable

Class is made dumpable by implementing interface org.jpc.emulator.SRDumpable and implementing method dumpSRPartial(org.jpc.emulator.SRDumper) and constructor <init>(org.jpc.emulator.SRLoader). Non-static inner classes can not be dumpable (make them static using tricks similar to what javac uses).

If dumped class has dumpable superclass, the first thing dumping function needs to do is to call dumper function of superclass and first thing loading constructor needs to do is to call loading constructor of superclass. If class has no dumpable superclass, dumper doesn't need to do anything special, while loader needs to call objectCreated(this) on SRLoader object passed as parameter.

Following these fixed parts, dump all members that are part of mutable state in emulator core.

9.1 Member dumping/loading functions

There is dumping/loading function for following (all functions dumping/loading reference types can handle null):

- boolean: SRDumper.dumpBoolean, SRLoader.loadBoolean
- byte: SRDumper.dumpByte, SRLoader.loadByte
- short: SRDumper.dumpShort, SRLoader.loadShort

- int: SRDumper.dumpInt, SRLoader.loadInt
- long: SRDumper.dumpLong, SRLoader.loadLong
- String: SRDumper.dumpString, SRLoader.loadString
- boolean[]: SRDumper.dumpArray, SRLoader.loadArrayBoolean
- byte[]: SRDumper.dumpArray, SRLoader.loadArrayByte
- short[]: SRDumper.dumpArray, SRLoader.loadArrayShort
- int[]: SRDumper.dumpArray, SRLoader.loadArrayInt
- long[]: SRDumper.dumpArray, SRLoader.loadArrayLong
- double[]: SRDumper.dumpArray, SRLoader.loadArrayDouble
- <dumpable type>: SRDumper.dumpObject, SRLoader.loadObject
- special object: SRDumper.specialObject, SRLoader.specialObject

9.1.1 Notes:

- Dumpable objects come out as type of org.jpc.emulator.SRDumpable.
- Special objects are various static objects that don't need to be stored because they don't have mutable fields.
- Don't dump fields related to event state feedback.
- Don't dump temporary flags that are only used while PC is running. Savestate when PC is running isn't possible anyway.
- Some connectors dump fields related to connector output, some don't.

10 Advanced: Writing event targets

Whereas output connectors are the way output is dispatched, input is dispatched via event targets. Event targets need to implement interface org.jpc.emulator.EventDispatchTarget.

Event targets also provode methods which then encode events and dispatch them forward (without doing anything else) to event recorder. Also, event targets may have methods for obtaining state.

10.1 Interface org.jpc.emulator.EventDispatchTarget

Interface that marks class capable of receiving events.

10.1.1 Method setEventRecorder(EventRecorder)

Set the event recorder input events are sent to.

10.1.2 Method startEventCheck()

Signals target to reset all state related to event checking and state feedback. This may be called at any time in order to reinitialialize event checking/feedback state.

10.1.3 Method doEvent(long, String[], int) throws IOException

Event dispatch handler. The first argument is event time, second is parameters and third is what to do with it. If target doesn't like the event, throw IOException. Following types (the integer parameter) are used:

```
0 (EventRecorder.EVENT_TIMED): Time has been assigned for event.
1 (EventRecorder.EVENT_STATE_EFFECT_FUTURE): Future event in event replay for reinitialization
2 (EventRecorder.EVENT_STATE_EFFECT): Past event in event replay reinitialization
3 (EventRecorder.EVENT_EXECUTE): This event occurs now. Execute the effect.
```

10.1.4 Method endEventCheock()

End event reinitialization. Usually unused.

10.1.5 Method getEventTimeLowBound(long, String[]) throws IOException

Return the time value that's the earliest possiblity for this event to occur. Returning any time in past (including -1) causes event to fire as soon as possible. The long parameter gives the current scheduled time for event.

11 Writing modules

Modules are various extensions that run inside emulator core. As such, they affect sync. Modules must implement interface org.jpc.emulator.HardwareComponent (they are hardware components) and must be dumpable. Additionally, they need either constructor <init>() or <init>(String). The first is if no parameters are passed, the second is for case where parameters are passed.

Aside of the constructors, modules need to obey the ordinary conventions for hardware components. No code outside modules needs to know that module exists.

12 Writing plugins

Plugins handle various UI tasks. They need to implement interface org.jpc.Plugin.

12.1 Interface org.jpc.pluginsbase.Plugin

12.1.1 Method systemShutdown()

Called when emulator shuts down. Either called in dedicated thread or in thread that called emulatorShutdown(). These handlers should do the bare minimum to get files on disk to consistent state. After these calls from all plugins have finished, emulator exits. Do not try to manipulate UI from these methods, as doing that easily leads into deadlock.

12.1.2 Method reconnect(PC)

Gives new PC to connect to. Null is passed if plugin should disconnect.

12.1.3 Method main()

Called in dedicated thread after plugin is initialized.

12.1.4 Method pcStopping()

Called after PC has stopped.

12.1.5 Method pcStarting()

Called before PC starts.

12.1.6 Method notifyArguments(String[])

Pass arguments from command line.

12.1.7 Constructor <init>(Plugins)

This constructor is used to initialize plugins that don't take parameters.

12.1.8 Constructor <init>(Plugins, String)

This constructor is used to initialize plugins that take parameters.

12.2 Class org.jpc.pluginsbase.Plugins

This class provodes various methods for manipulating plugins.

12.2.1 Method isShuttingDown()

Returns true if Plugins.shutdownEmulator() has been called somehow, either via VM exit, CTRL+C or explicitly. Useful to skip cleanups involving GUI, as these are too deadlock-prone.

12.2.2 Method shutdownEmulator()

Shut down and exit the emulator. All plugin shutdown functions are called in this thread.

12.2.3 Method reconnectPC(PC)

Signal reconnectPC event to all plugins.

12.2.4 Method pcStarted()

Signal pcStarting() event to all plugins.

12.2.5 Method pcStopped()

Signal pcStopping() event to all plugins.

13 Inter-plugin communication

13.1 Receiving communications

To receive invocation/call by name 'foo-bar', declare public method named 'eci_foo_bar'. Arguments to this method can currently be String, Integer (int) or Long (long). Last argument may be array over these types to get variable number of arguments. On call, each argument gets value from call. If last argument is array, it gets all overflowing arguments. If return type is void or method returns boolean false, call is assumed to have completed. If return value is boolean true, it is assumed that there is more processing.

13.2 void org.jpc.pluginsbase.Plugins.invokeExternalCommand(String cmd, Object[] args)

Invoke command asynchronously, broadcasting to all plugins. Does not wait for slow commands to complete. cmd is the name to send and args are the arguments to pass.

13.3 void org.jpc.pluginsbase.Plugins.invokeExternalCommandSynchronous(String cmd, Object[] args)

Same as invokeExternalCommand, but waits for slow commands to complete.

13.4 Object[] org.jpc.pluginsbase.Plugins.invokeExternalCommandReturn(String cmd, Object[] args)

Similar to invokeExternalCommandSynchornous, but:

- Quits calling more plugins when it gets successful reply.
- Returns said reply

13.5 void org.jpc.pluginsbase.Plugins.returnValue(Object... ret)

Gives return value to return from call and signals that command has completed.

13.6 void org.jpc.pluginsbase.Plugins.signalCommandCompletion()

Signals that command has completed. Only needed if there is no return value and eci_ method returned false (not done yet).

14 Lua kernel programming

At startup, kernel gets its arguments in 'args' table and the script name to run in 'scriptname' string. It should enter the named script in protected mode.

The Lua VM exports numerious callbacks to kernel. The kernel can then choose to omit, wrap or re-export these to Lua scripts.

- Always grab any functions used into local variables so nobody can mess with them
- Don't use global variables in kernel (except for those passed).