Manual for jPSXdec: PlayStation 1 Media Decoder/ Converter written in Java

Version 1.00

Michael Sabin jpsxdec@gmail.com

Table of Contents

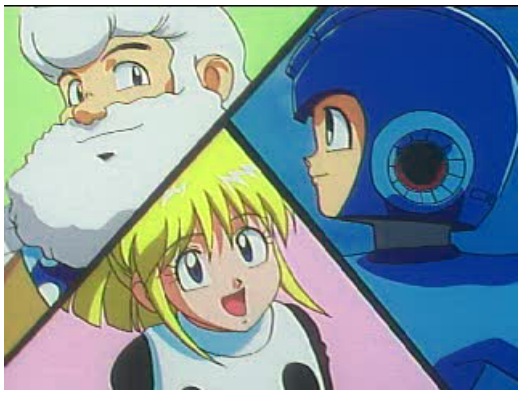
1 About

jPSXdec is a cross-platform PlayStation 1 media reader, decoder, and converter developed using the Java framework.

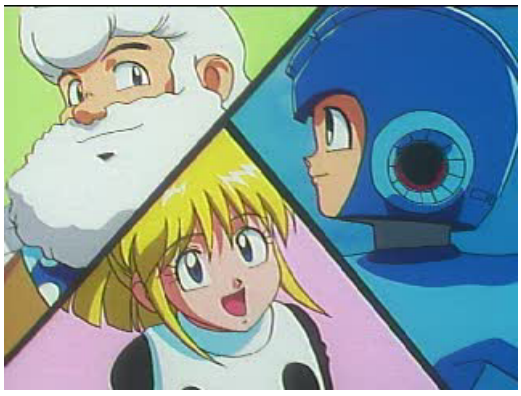
1.1 Quality

When the first PlayStation console was in its prime, several hackers developed tools to interact with the game data, such as emulators and media rippers. Unfortunately nearly all of them got something wrong: the video colors.

Wrong colors



Right colors



Notice on the left that the wrong colors make the yellow and green darker? On the right are the correct colors with the yellow and blue being lighter. jPSXdec not only gets these colors right, but its entire decoding process is designed to maximize the quality of every pixel.

2 Minimum system requirements

jPSXdec is almost universally cross-platform, running on Windows, Mac, and Linux, but requires Java to be installed to work.

There are several Java Virtual Machines and frameworks available. jPSXdec has been primarily tested on Oracle's Java. It requires Java version 6 or higher, although it is recommended to use never versions if available.

3 File and disc image formats recognized by jPSXdec

jPSXdec is primarily designed to work with CD disc images. The supported formats include

- ISO .iso images (2048 bytes/sector)

- the .bin file of BIN/CUE images (2352 bytes/sector or 2448 bytes/sector) jPSXdec does not use the .cue file
- Alcohol 120% .mdf files (2448 bytes/sector)

It is **strongly** recommended that you use the BIN/CUE image format. Important data is lost in ISO disc images. Full disc ripping to the BIN/CUE format can be done using various tools on each platform.

jPSXdec can also usually read extracted .str video and .xa audio files because those files are simply extracted portions of BIN/CUE disc images. You may also process TIM images from the command-line using the `-static` command (see section 6.4).

4 Supported media formats

PlayStation games used countless ways to encode their audio, video, images, 3d models, fonts, text, animations, and all other data. There's absolutely no way to know how most games encoded most formats.

jPSXdec is focused on finding and extracting only the **audio** and **video** clips found in games. It can identify the most common formats. Special handling is added on a game-by-game basis to support unique ways of encoding audio and video.

Below are the formats supported by jPSXdec.

4.1 Audio commonly referred to as “XA” audio

XA stands for "eXtended Architecture" which is a CD format specification that includes a standard way to encode audio. The audio is a unique variation of adaptive differential pulse-code modulation (ADPCM) audio. This kind of audio is found in most games. This audio format is also used by the Phillips CD-i console, so jPSXdec may be able to detect and decode CD-i game audio as well.

4.2 SPU audio commonly referred to as “VAG” audio

VAG stands for “Very Audio Good” (an “Engrish” term obviously). VAG is a file format that contains audio data designed specifically for the PlayStation’s onboard Sound Processing Unit (SPU). The SPU is used to decode ADPCM audio that is slightly different from the XA ADPCM audio format. SPU audio can exist outside of VAG files, so I usually refer to this audio as “SPU” audio.

Some PlayStation game developers chose to use SPU audio instead of XA audio data for their videos. jPSXdec contains special handling for some game videos that use this audio format.

4.3 “STR” full motion videos

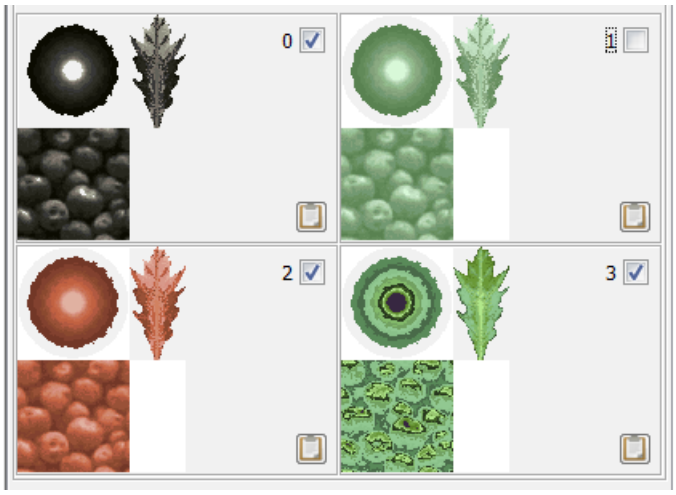
STR stands for "stream," meaning a video stream which is often combined with XA audio. The PlayStation's onboard "MDEC" chip (which stands for "Motion Decoder") is used in the

process of decoding these videos.

4.4 “TIM” image format

While jPSXdec is primarily focused on audio and video, it does make one exception for a popular image format used in most games. This format is called "TIM" (I don't know why it's called TIM).

The TIM image format may confuse those who are unfamiliar with it. Most TIM images contain multiple "palettes," also known as "Color LookUp Tables" (CLUT). Each palette can be applied to the image data, generating a unique image. Here is an example of a single TIM image with four palettes.



4.5 Files

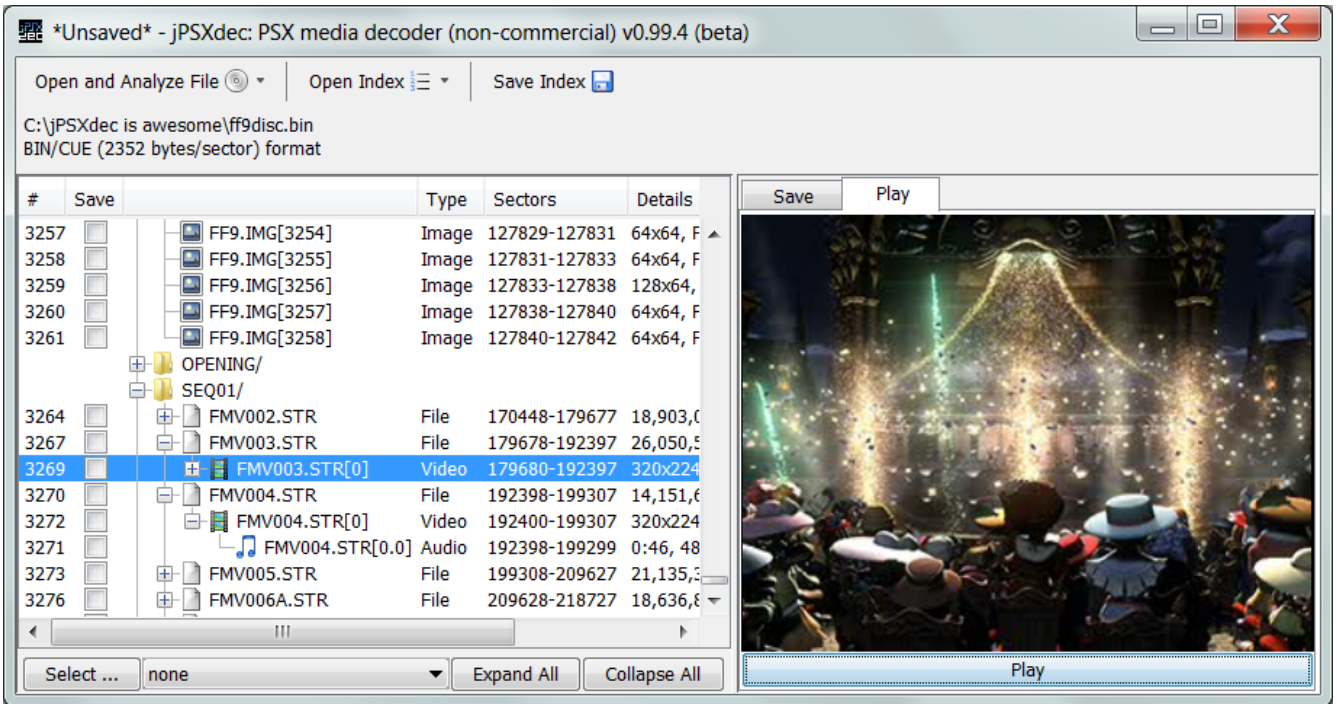
These are normal files you would see on the disc as if it was inserted in your CD drive (as specified by the ISO 9660 standard). Please see important information about how files are stored on PlayStation game discs in section 7.4.

5 GUI (Graphical User Interface)

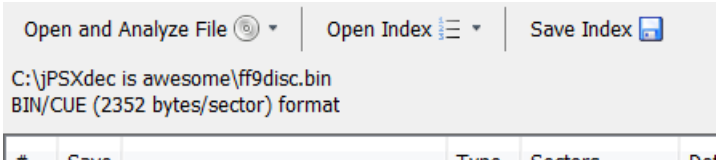
Starting jPSXdec without arguments will launch the graphical user interface.

On Windows, double-click `jpsxdec.exe`. On Mac and Linux, double-click `jpsxdec.jar`.

If there is only one argument supplied, jPSXdec will auto-detect if it is an index file or a disc image and immediately open it in the GUI.

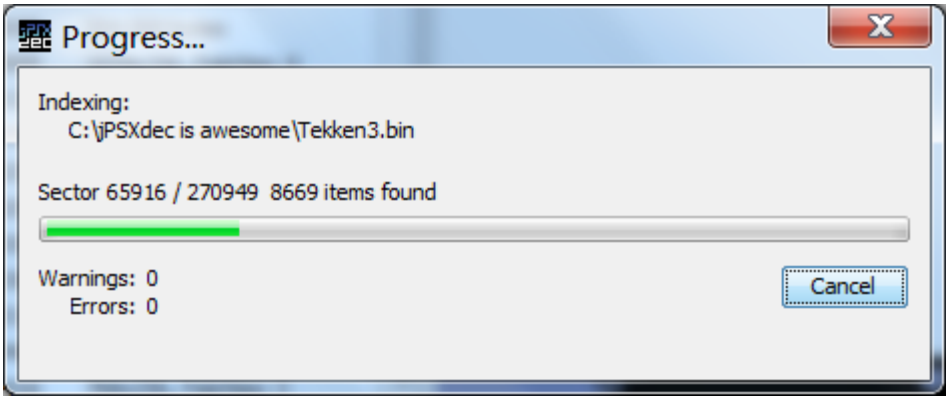


5.1 Toolbar



5.1.1 Open and Analyze File

A file dialog box will let you select a CD disc image or other supported file format to analyze (see chapter 3 for supported formats). Or alternatively, a drop-down list with the most recent files opened. The analysis begins immediately.



5.1.2 Open Index

A file dialog box will let you select a previously saved disc index file. Or alternatively, a drop-down list with the most recent files opened. Chapter 9 goes into more detail about index files.

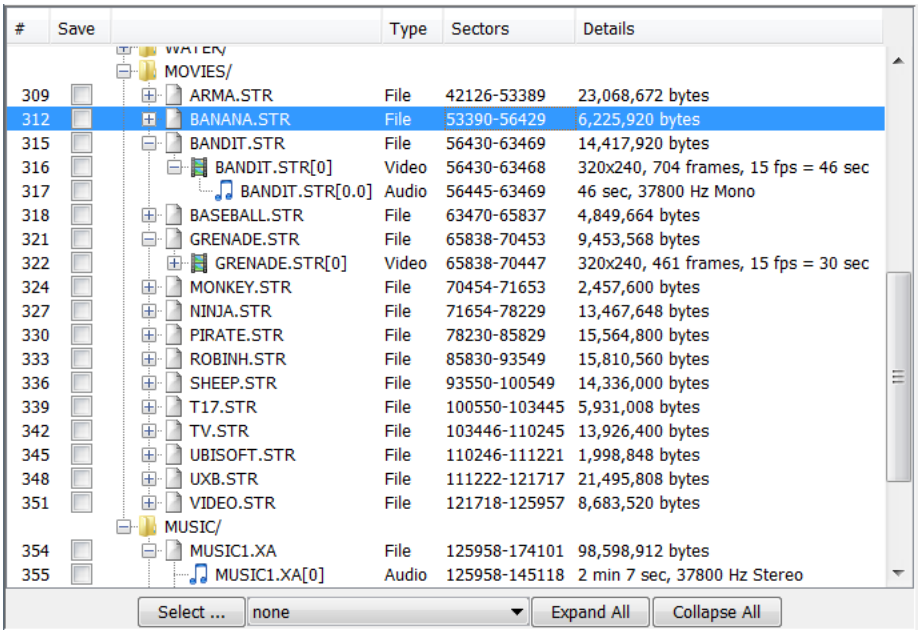
5.1.3 Save Index

After opening and analyzing a disc image, you can save the index it generates so you don't have to re-analyze the disc image. Chapter 9 goes into more detail about index files.

5.1.4 Disc info

Under the toolbar buttons is a little information about the disc image: the file name, and the format of the disc.

5.2 List of items



5.2.1 Columns

#:	Unique number associated with the item
Save:	Toggle this check box to include for saving
Type:	Type of item
Sectors:	The disc sectors that the item spans (inclusive)
Details:	More information about the item

5.2.2 Buttons

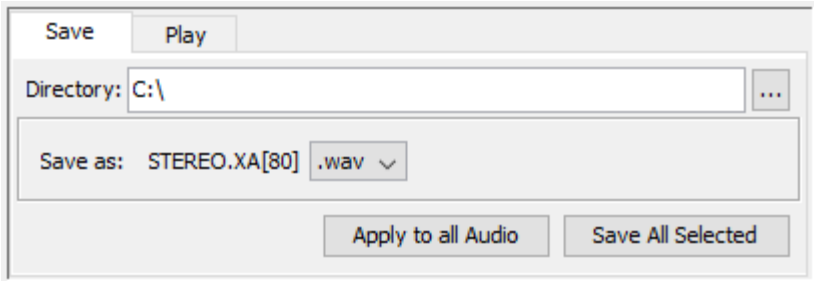
Select...: Pick an item type in the drop-down menu, then press this button to select (check) all the items of that type.

Expand All/Collapse All: Fully expand or collapse all tree nodes.

5.3 Save panel

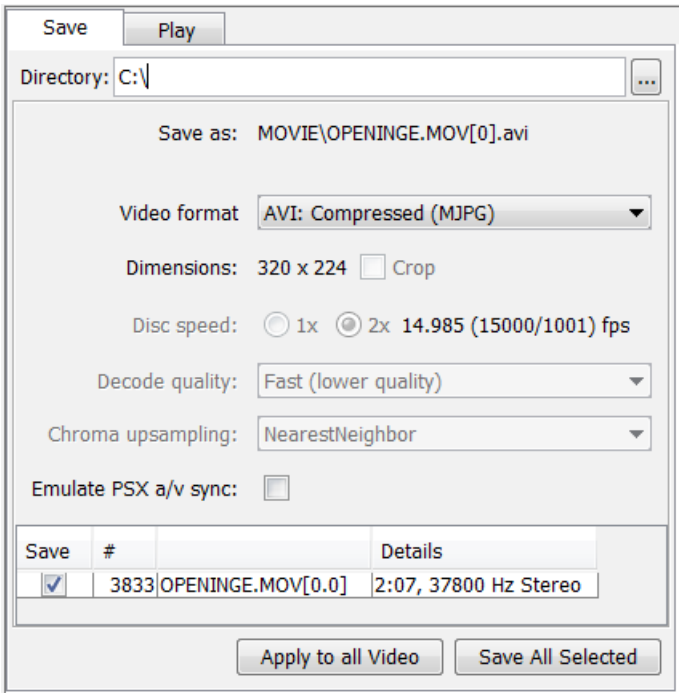
At the top of the "Save" panel is the "Directory" where files will be saved to. Please note that if source data exists within a directory on the disc, when that data is extracted, **it will recreate the game's directory structure inside the output directory**. This is necessary to avoid the possibility of multiple items of the same name overwriting one another.

5.3.1 Audio save panel



Save as: Shows the output file name and lets you select the output format. See section 7.1 for details about the audio format options.

5.3.2 Video save panel



Save as: The file name or names that will be created.

Video format: Format to save the video. See section 7.2 for details about these formats.

Dimensions, Crop: The dimensions of the video that will be saved. The "Crop" checkbox will be enabled in cases where the video can be uncropped. See section 7.2.2 for details about this case.

Disc speed, fps: The frame rate or "frames-per-second" of the video. Some videos have an ambiguous frame rate, so manually setting the "Disc speed" may be necessary. See section 7.2.1 for details about this case.

Decode quality: The quality of the video decoder. See section 7.2 for details about these options.

Chroma upsampling: Only applicable to highest RGB quality decoding. In order to decode images to RGB, the color (chroma) information has to be scaled up (i.e. upsampled). This option sets the scaling (i.e. interpolation) method. See section 7.2.3 for the full explanation of what this means.

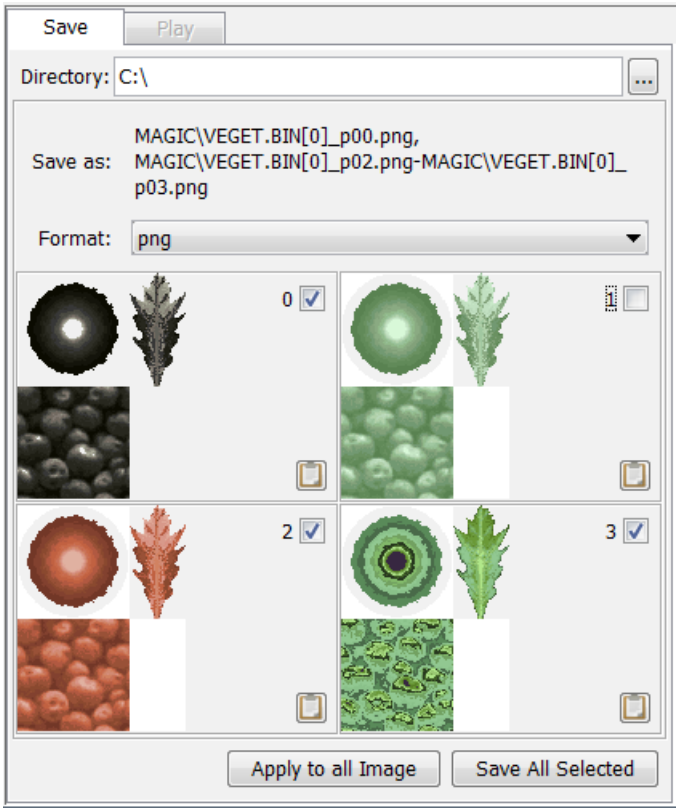
Emulate PSX a/v sync: This synchronizes the audio and video like they might appear on the actual PlayStation hardware. See section 7.2.1 for more details. **In most cases you wouldn't want to use this option.**

List of audio streams (if available):

Some movies don't have audio. Movies with audio almost always have only one track. The only time I've seen a movie with multiple audio tracks was due to disc

corruption. This list lets you select which audio tracks to use. If there are more than one track, when you select one, it will automatically deselect other overlapping audio tracks.

5.3.3 Images ("TIM" images)



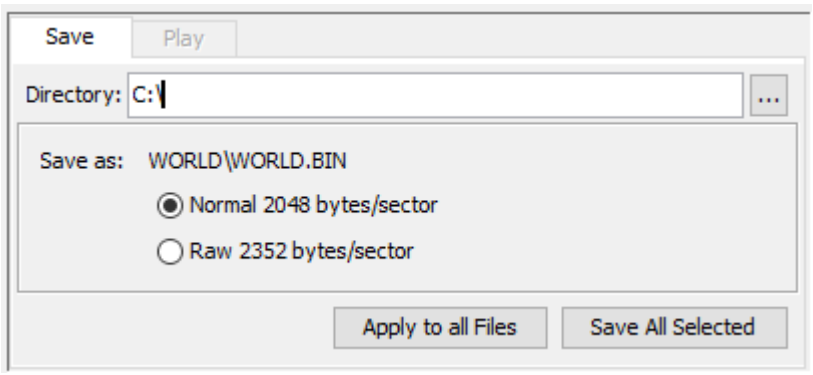
Save as: Summary of files that will be generated.

Format: The format to save the image(s) as. See section 7.3 for more information about the image format options.

(palettes): TIM images will have one or more palettes, or "Color LookUp Tables" (CLUT). You can check which palettes you want to save. A separate image will be saved for each palette. See sections 4.4 and 7.3 for more details.

Click the clipboard button to copy that palette image to the clipboard.

5.3.4 Files



Save as: Shows the output path and file name.

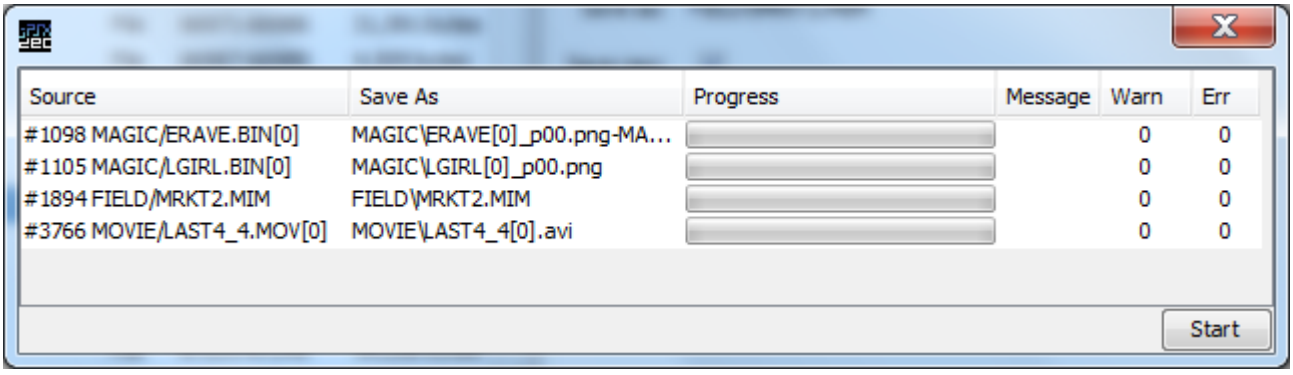
(format): "Normal" option will extract the file data as if you copied the file off the disc using normal methods (no raw sector headers). Using the "Raw" option will extract the data with the raw sector headers. "Normal" will be disabled if the file contains data that requires the raw sector headers. The "Raw" option will be disabled if the source disc image does not contain raw sector headers. **See section 7.4 for more important information about extracting files.**

5.3.5 Apply to all <type>

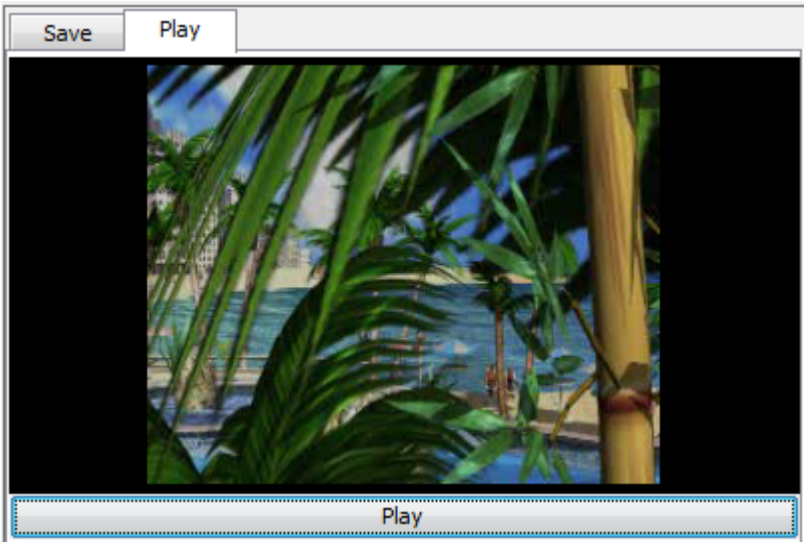
Pressing this button will apply most of the settings of the current item to all other items of that type. For example: sets all videos to be saved with high quality options. This may not always result in the desired settings. If not, change the settings manually.

5.3.6 Save All Selected

This will show the dialog with all the items selected (checked) for saving. Pressing Start will begin the saving process. Please note that **it will overwrite any existing files**. Also if source data exists within a directory on the disc, when that data is extracted, **it will recreate the game's directory structure inside the output directory**. This is necessary to avoid the possibility of multiple items of the same name overwriting one another.



5.4 Play panel



Lets you play video and audio straight off the source file. There's nothing displayed when playing audio.

5.5 jpsxdex.ini settings

The GUI saves the file jpsxdec.ini in the startup directory, or working directory, so it can maintain the history of previously opened files and folders.

6 Command-line

6.1 Quick start

First, the source file (whether it be a disc image or a single media file) needs to be analyzed and an index generated.

Example to generate an index:

```
java -jar jpsxdec.jar -f MyGameImage.bin -x MyGameImage.idx
```

Once the index is created, open it to review what was found in the source file.

Identify the id-number of an item you are interested in, and run the command to extract/decode it.

Example to extract the first (#0) item with default options:

```
java -jar jpsxdec.jar -x MyGameImage.idx -i 0
```

6.2 Main command-line options

```
java -jar jpsxdec.jar -help/-h/-?
```

Displays main help.

```
java -jar jpsxdec.jar -f <in_file> -x <index_file>
```

Build an index of `<in_file>` and save it as `<index_file>`.

```
java -jar jpsxdec.jar [-f <in_file>] [-x <index_file>] <command_and_options>
```

Perform a command that requires an index. Either the `<in_file>` or `<index_file>` need to be specified, or both.
If only the `<in_file>` is specified, then it is immediately indexed, but the index is not saved. This is useful for small .str or .xa files that only contain one or two audio or video streams.
If only the `<index_file>` is specified, then jPSXdec uses the source file stored in the index header as the `<in_file>`.
If both the `<in_file>` and `<index_file>` are specified, then jPSXdec uses the index, but uses the `<in_file>` specified on the command-line instead of the file defined in the index.

The available `<command_and_options>`:

```
-item/-i # [-?/-h/help] [-dir <directory>] [item_command_options]
```

The `#` can be the item's number found in the index, or it can be the ID found in the index file.

If `-?` or `-h` or `-help` is used, displays the options available for item `#`.

Otherwise processes item `#` using optional `[item_command_options]`. See each item's help for detailed options.

Optional `<directory>` specifies the output directory where the output files will be saved. Default is the current directory. Please note that **it will overwrite any existing files**. Also if source data exists within a directory on the disc, when that data is extracted, **it will recreate the game's directory structure inside the output directory**. This is necessary to avoid the possibility of multiple items of the same name overwriting one another.

```
-a <type> [-dir <directory>] [item_command_options]
```

Process all disc items of type `audio`, `video`, `image`, or `file`. It will try to apply the `[item_command_options]` to each item, but may not always be able to. If it doesn't, process each item individually.

Optional `<directory>` specifies the output directory where the output files will be saved. Default is the current directory. Please note that **it will overwrite any existing files**. Also if source data exists within a directory on the disc, when that data is extracted, **it will recreate the game's directory structure inside the output directory**. This is necessary to avoid the possibility of multiple items of the same name overwriting one another.

`-visualize <output_pdf>`

Creates a massive pdf representing the physical layout the disc with color-coded sectors and index items (for debugging).

6.3 Options for extracting disc item types

See chapter 4 for more information about these types.

6.3.1 STR Video

For about the highest quality output, use these flags:

```
-quality high -vf avi:rgb -up Lanczos3
```

See details about output formats in section 7.2.

Command-line options

`-quality/-q <low, high, psx>`

Selects the decoding quality. Default is `high`. Option only applies to `avi:rgb`, `bmp`, and `png` formats.

`low` quality is fast and uses integer based arithmetic.

`high` is slower, but should create better output because it uses floating point arithmetic (although it may not be noticeable without close inspection). It lets you choose scaling (i.e. interpolation) method for chroma (color) upsampling for a noticeable quality improvement. Use `-up` option to set interpolation method.

There really isn't any reason to use anything but high.

`psx` quality tries to duplicate what the PlayStation would generate (about the same quality as `low`). It's not perfect, but is currently better than any other decoder ever written.

`-vidfmt/-vf <format>`

Selects the output format. Default is `avi:mjpg`. See section 7.2 for details about these formats.

`avi:mjpg` – Smallest output with only a small loss of quality.

`avi:rgb` – Completely uncompressed, RGB (device independent bitmap, i.e. DIB) AVI.

Very large, but with the best quality.

`avi:yuv` – Some quality loss, but smaller than `avi:rgb`. Uses the fourcc YV12 codec (also known as rec.601 colors).

`avi:jyuv` – Most similar to PlayStation color spectrum. Uses the fourcc YV12 codec, but pixel values use the full [0-255] range (sometimes called "pc.601" or jpeg color range).

`png`, `jpg`, `bmp` – Saves a series of images.

`bs` – Demuxed compressed bitstream frame data found in separate sectors on the disc. 'Demuxing' is the first step in the decoding process, which simply combines the separate frame chunks into a contiguous block.

`mdec` – Generated from the demuxed bitstream data. This data would then be fed into the PlayStation MDEC chip, which would produce viewable images.

`-start <, -end`

Lets you specify a frame to start decoding, end decoding, or both. Audio is ignored when using these option because accurate audio decoding must begin from the start of the audio stream (you can't start in the middle). Frames can be defined using any combination of formats as described in chapter 8 (e.g. `#15.4` or `@668674.2`).

`-up <upsampling>`

Specifies the chroma upsampling interpolation method. See section 7.2.3 for details about why this option is available. Default is `Bicubic`. Ignored if not using `-q high`. Options (not case-sensitive): `NearestNeighbor`, `Bilinear`, `Bicubic`, `Bell`, `Mitchell`, `BSpline`, `Lanczos3`, `Hermite`.

`-noaud`

Don't decode audio with the video. Ignored when video does not have audio, or when not saving as AVI.

`-psxav`

By default, when saving the audio and video as an AVI, they are set to start playing at precisely the same time. While that is probably how the video was intended to be seen, that is not exactly how it is played back on the PlayStation. The audio most likely starts where it first appears on the disc, which could be up to 0.15 seconds after the video starts. This option saves the audio and video to begin exactly when they appear on the disc (option ignored if not saving AVI with audio).

`-nocrop`

PlayStation 1 movies technically must have dimensions that are multiples of 16. In cases where games need movies that do not have such dimensions, the extra space is filled with garbage and is cropped off before being drawn on screen. When this option is used, that extra data is not cropped. Option ignored if saving as bitstream (`bs`) or `mdec` format.

`-ds <1 or 2>`

"Disc speed" Movie frame rates are entirely dependent on the speed the disc spins. PlayStation 1 can spin discs at 1x speed, or 2x speed. Most games use 2x speed for all their movies. jPSXdec can usually automatically detect the disc during indexing, but if the speed is unclear, it will assume 2x. In the rare case that is incorrect, this option lets you force using 1x disc speed instead (effectively halving the frame-rate). Option ignored if not saving as AVI.

`-num <#, @, index, sector, header>`

When saving a sequence of images, indicates how to number the frame files. See chapter 8 for details about these options. `index` will use the frame index (starting as 0), `sector` (`@`) will use the start sector, `header` (`#`) will use the frame number found in sector headers. Default is `index`. Option is ignored when saving as AVI.

`-frameinfodump`

Prints out detailed information about each video frame (for debugging).

`-replaceframes <str_replace_xml>`

Replaces frames of a video item according to the options in the xml file. See chapter 10 for details.

6.3.2 XA Audio & PlayStation Sound Processing Unit (SPU) Audio

Standard XA ADPCM audio used by Sony PlayStation 1 and Phillips CD-i, and ADPCM audio decoded by the PlayStation's Sound Processing Unit (SPU). See chapter 4 for details.

Command-line options

`-vol <range of 0 to 100>`

This does more than just adjust the volume. Because it does the adjustment before clamping the waveform and rounding it to an integer, it's able to keep data that would have been lost otherwise. This can also help in cases where a game's audio has distortions due to the clamping. Adjusting the volume at this point should produce better quality than adjusting the volume later using another program. Default is 100.

`-audfmt/-af <format>`

Output audio format. Possible options are `wav`, `aif`, `au`, and `snd`. Default is `wav`.

`-replacexa <audio_file>`

(Only for XA audio) Converts an audio file into XA ADPCM format and overwrites the XA ADPCM audio stream. The audio file properties must match XA audio properties (channels, sample rate, etc.) and be of the same length. Accepted audio formats are `wav`, `aif`, `au`, and `snd`. This operation cannot be undone, so backup your disc before running it.

`-replacexa <other_index_file> -xa <item#>`

(Only for XA audio) Overwrites the XA ADPCM audio stream with XA data from another file. The XA streams must have the same format and length. This operation cannot be undone, so backup your disc before running it.

Example:

If you have `GAME.bin`, start by generating the index for it (e.g. `GAME.idx`). Identify the XA audio stream you want to overwrite in the index (e.g. #23).

Now if you have new XA audio in a file called `NEW.xa`, generate the index for it as well (e.g. `NEW.idx`). Identify the item number of the new audio stream in `NEW.idx` (e.g. #1). Now you have what you need to replace the audio.

```
java -jar jpsxdec.jar -x GAME.idx -i 23 -replacexa NEW.idx -xa 1
```

6.3.3 TIM image

TIM images regularly used in PlayStation games.

Command-line options

`-imgfmt/-if <image format>`
: Format to save the TIM image. Use the item's help for available options. These options will be different depending on if the TIM image has a palette. Default is `png`.

`-pal <palettes to save>`
: TIM images may come with one or more palettes. By default, all palettes are saved as a separate output files. This option lets you select exactly which palettes to save using a comma delimited list, or hyphens (no spaces). For example, to save the first and third, fourth and fifth palettes, you could use this: `-pal 1,3-5`

`-replacetim <tim_file>`
: Replaces an existing TIM image with a new TIM image. The new TIM image must exactly match all properties of the existing TIM image.

6.3.4 File

Normal files on disc images are detected and can be extracted and saved. Note that jPSXdec ignores the exact file size and just saves the full sectors.

Command-line options

`-raw`
: By default files will be saved using the normal format that you would get if you copied the file off the disc using normal methods (no raw sector headers). Using this raw option will include the raw sector headers if the source disc image contains raw headers. This raw option will be ignored if the file contains data that requires it to include the raw sector headers. See item's help for if it is available. **See also section 7.4 for more important information about extracting files.**

6.4 Operations not requiring an index

```
java -jar jpsxdec.jar -f <in_file> <in_file_command> [in_file_command_options]
```

Perform an operation directly on `<in_file>`, not needing an index file.

`-static <type>`
: Process special static file types

  `tim`
  : Converts a tim image to .png, one image per palette.

`bs` or `mdec`

Converts a single bitstream or mdec frame to an image.

`-dim` *width*`x`*hight*

Specifies the dimensions of the frame, width and height separated by an 'x'. Required.

`-quality/-q <low, high, psx>`

Decoding quality (default `high`). See section 7.2 for details about these options.

`-fmt <png, bmp, jpg, mdec>`

Output format. `mdec` only available for bitstream sources. Default `png`.

`-up <upsampling>`

Chroma upsampling interpolation method for `high` quality. Default `Bicubic`. See section 7.2.3 for details.

`-debug`

Prints the full conversion details (very low level and technical). Requires the Java `-ea` (enable assertion) flag (for debugging).

`-copysect <start_sector>-<end_sector>`

Copy the sectors from start to end (inclusive) separated by a hyphen, to a file named `<in_file_name><start_sector>-<end_sector>.dat`

`-sectordump <out_file>`

Write a list of sector types to `<out_file>` (for debugging).

7 Output saving formats considerations

Each media format supported by jPSXdec can be exported and saved in various formats using various methods. Here we explore the pros and cons of each option.

Please note that if source data exists within a directory on the disc, when that data is extracted, **it will recreate the game's directory structure inside the output directory**. This is necessary to avoid the possibility of multiple items of the same name overwriting one another.

7.1 Audio

wav audio files are universal. The other formats (aiff, au, aifc, snd) are only offered because the Java platform supports them for free. You probably won't have any use for them.

7.2 Video output formats and quality differences

jPSXdec offers several options and formats to save videos. Each has pros and cons.

7.2.1 Video frame rates (frames-per-second or "fps") in output AVI

Frame rates only apply when saving as AVI videos. When saving videos as an image sequence, frame rate is obviously ignored and each frame is saved as a separate image.

The PlayStation doesn't require videos to have a consistent frame rate. Some games have videos with variable frame rates, and most games have movies where the frame rate isn't precisely as detected by jPSXdec (but the variance isn't perceivable).

When saving as AVI, jPSXdec tries to place the frames in the AVI time-line as close to where they belong. This may mean that duplicate frames are inserted to keep the frames in sync with the audio. AVI files have a feature so duplicating a frame adds almost no extra size to files, **but this feature is not supported by all video players** (unsupported players will skip duplicate frames, causing the video the play extra fast). Silent audio may also be inserted in cases where the audio is out of sync.

One important factor when detecting the frame rate is the speed the disc spins. There are two speeds the PlayStation can spin the disc: 1x and 2x. The frame rate is twice as fast when the disc spins at 2x speed. Usually jPSXdec can detect the speed the disc wound spin when it played the video. When it can't, you are given the option to select the disc speed. You can evaluate for yourself which speed is right for the video.

Another option is offered to emulate how the audio and video would be synchronized as it might look like on the actual PlayStation. The PlayStation hardware doesn't start the audio and video at the exact same time. Enabling this will add some delay to the audio or video so they are in sync like how they would be on the PlayStation. **In most cases you shouldn't use this.**

7.2.2 Cropping

PlayStation 1 movies technically must have dimensions that are multiples of 16. In cases where games need movies that do not have such dimensions, the extra space is filled with garbage and is cropped off before being drawn on screen. If you'd like to see the part that is cropped off, use the option to disable cropping.

This option is provided mostly for curiosity's sake, but it does have some value when replacing videos.

This option does not apply to the bitstream or "mdec" output formats.

7.2.3 bmp or png image sequences, and AVI uncompressed RGB (a.k.a. device independent bitmap, or DIB) and video quality

The video quality options are only available for these formats.

These formats generate the largest output because they completely uncompress the video.

Here I would like to emphasize that *there is no other program that can decode video more accurately than jPSXdec*. Therefore, if you really want the pixels to have the most accurate colors, use this method.

jPSXdec offers three quality options when converting to these outputs. But in short, **you should always use high quality**.

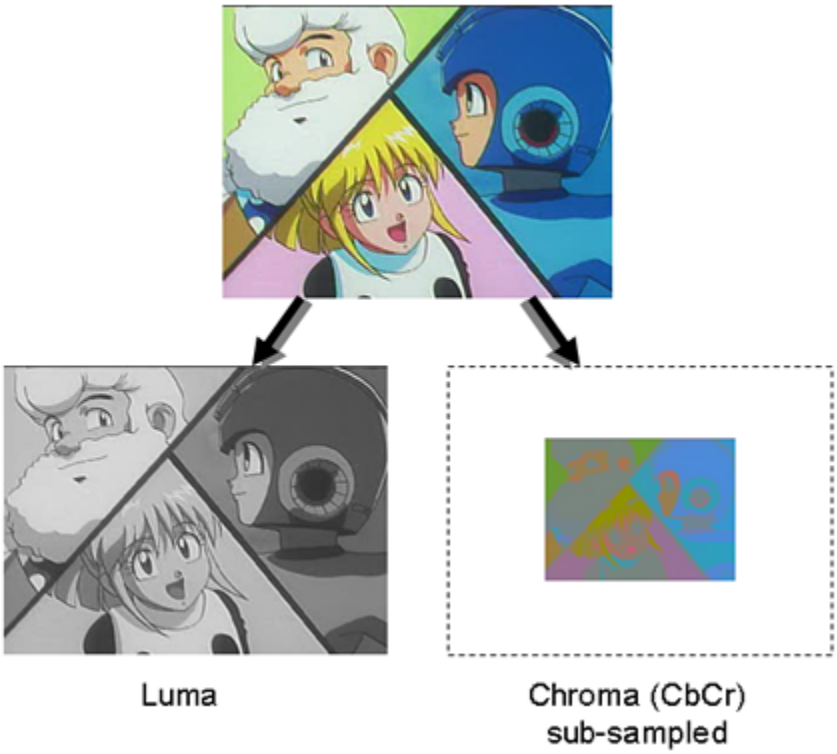
- **high** high is the slowest of the three, but should create better output because it uses floating point arithmetic and allows you to choose the chroma up-scaling method for a noticeable quality improvement (see below for details).
- **low** low quality is a little faster than the others and uses integer based arithmetic. Nearest-neighbor is used for chroma up-sampling (see below). This is *only* used for the real time video player because it's the fastest.
- **psx** psx quality tries to duplicate what the PlayStation would generate (about the same quality as low). It's not perfect, but is currently better than any other decoder ever written. The "psx" option is really only useful for academic purposes: how close can we get to actual hardware output?

Again, **you should always use high.** It's as simple as that. The low quality might be faster, but who cares? Take the few seconds more and get the better output.

Chroma up-sampling

Part of the decoding requires scaling up (or up-sampling) the color (or "chroma") component of the image. Encoded video frames are separated into two components: the brightness (i.e. "luma") and the colors (i.e. "chroma"). The color component is scaled down (i.e. "sub-sampled") to help compress the image.

When decoding, the color component needs to then be scaled up (i.e. "up-sampled"). The methods used to upscale the colors are the same methods you can find to scale an image in any good image editor such as Photoshop or Gimp (e.g. nearest neighbor or bicubic). No scaling method is perfect, but some are usually better than others. In general, bicubic is usually a good choice. jPSXdec provides all of the most common methods to upscale the color component for you.



With all this talk of high quality being the best, I should add one caveat. If you're interested in applying video enhancing filters to the video, RGB images may not always be the best format to use. For example, Avisynth can apply enhancing filters (e.g. noise reduction) to YUV or MJPG formats better than RGB. Here comes the trade-off: do you want the most accurate colors, or do you want the filters to look the best? (I would choose the filter since the filter effect will change the colors anyway)

7.2.4 JPG and AVI with MJPG codec

The PlayStation video format is quite similar to the JPEG format. It's so similar that it is possible to translate *most* of the data directly from one format to the other (there is still a little quality loss), without the need to fully decode and then re-encode. There are some caveats however.

- Since this is mostly just a translation, the real decoding will be left to a JPEG decoder. Unfortunately no two JPEG decoders are alike. The exact decoding process the PlayStation uses is not going to be found anywhere besides jPSXdec. This means all other JPEG decoders will output slightly different colors compared to jPSXdec.
- There is still a small amount of quality loss during the translation due do some rounding. However, every other video decoder I've seen also throws away that rounding data, so translating to JPEG should be at least as high quality as any other decoder, but will not be accurate to the PlayStation method.

7.2.5 AVI YUV* (a.k.a. Rec.601) – AVI with YV12 codec

About 50% smaller than RGB. Unfortunately there are 2 major issues with this format.

- Rec.601 only uses a portion of the available 255 range to store data: [16-235] for Luma (Y), and [16-240] for Chroma (CbCr). The PlayStation's YCbCr format uses the full [0-255] possible range of values. jPSXdec scales the values to fit within the subset, resulting in some quality loss.
- You're left to the mercy of other tools to convert to RGB. Unfortunately all other tools (except for Avisynth) will convert YCbCr → RGB with a ½ pixel "chroma shift." This will cause incorrect colors in the output.

* Technically it's "YCbCr" not YUV

7.2.6 AVI "JYUV" (a.k.a. pc.601) – also using YV12 codec

This is like the normal AVI YUV format above, except keeps the full [0-255] range of possible values. It's sometimes called "JYUV" because JPEG also uses the full range. This is most similar to the original PlayStation video colors. Avisynth and ffmepg are the only tools I'm aware of that can handle this specific format. Again, 50% smaller than RGB, but all other tools (except Avisynth) will introduce a chroma shift when converting to RGB.

7.2.7 demuxed bitstream

PlayStation video frames are broken up into multiple chunks. "Demuxing" is the first step in the decoding process, which simply combines the separate chunks into a contiguous block. This contiguous block is the compressed bitstream of the image. Almost no one has any use for this format.

7.2.8 mdec

The second step in the decoding process is uncompressing the demuxed bitstream. This data would then be fed into the PlayStation MDEC chip, which would produce viewable images. Less than no one has any use for this format.

7.3 TIM images

The TIM image format is fairly well documented. The way transparency works in TIM files is a little odd, but reflects how the PlayStation hardware works. jPSXdec handles the colors and transparency differently than other tools, so you may find something new from it.

Output options are .bmp and .png for 24bit TIM images, and .bmp, .png, or .gif for paletted images.

7.4 Files

jPSXdec can detect 4 different styles of disc images as described in chapter 3. By default,

files from disc images are copied (almost) as if it had been normally copied off a disc by your computer. The raw option will include the raw sector headers. Files saved with raw sector headers are also wrapped in a "RIFF" header that helps other programs identify the data as having raw sector headers. Note that jPSXdec ignores the exact file size and just saves the full sectors.

WARNING Files, as we are familiar with them, are ignored by most PlayStation games. It was faster to hard-code sectors that contain data directly in the software than to spend a cycle reading the filesystem table before then reading the data. Due to this, in some cases data may extend outside the file boundaries, and even disc boundaries. You might consider extracting the actual sectors of interest instead of relying on the filesystem to determine the sectors to extract.

8 Frame numbering and look-up

There are a few places in jPSXdec where it is necessary to identify a frame in a video.

- The video command-line `-start` or `-end` options (input)
- The `-replaceframes` XML (input)
- The file name when saving a sequence of images (output)

jPSXdec supports referring to frames in 3 different ways, each with different pros and cons.

	Syntax	Pro	Con

Index (i.e. first frame is 0, second frame is 1, etc).
This is the default.
	Just the index number, e.g.
	`0`
	`5`
	`273`
		Simple, always starts at 0.	May not be consistent between disc images if there are corrupted sectors.

Frame number found in the sector headers.	Frame number prefixed by #. May also have a decimal number indicating a duplicate frame number, e.g.
	`#1`
	`#7`
	`#26.3`
		Always consistent between disc images.	Some videos don't store a frame number in the header. Some videos have duplicate numbers in the headers. Some videos first frame is not 1.

Sector where the frame starts.	Sector number prefixed by @. May also have a decimal number indicating more than one frame starting at the sector, e.g.
	`@11235`
	`@403749`
	`@89553.2`
		Always consistent between disc images.	Kind of strange, but is even more reliable than using the frame number in the sector headers.

It is very unlikely you would ever need to use anything but the default (index).

9 Index files

Index files use the extension .idx and contain important information about the contents of a disc image or media file. jPSXdec needs the index before it can perform any operations on a source file.

Index files may be edited by hand. If there's something wrong, jPSXdec will let you know.

The first line of index files contain a header indicating the jPSXdec version used to save the file. jPSXdec won't open index files saved with a different version.

Index files contain a path to the source file that the index was created from which jPSXdec will use to find the disc image (this can be overridden from the command-line).

All audio and video streams are listed independently, but jPSXdec will detect if an audio stream is associated with a video stream.

10 Replacing movies

Using the `-replaceframes` option on video items, jPSXdec can re-encode and replace video frames within streaming movies on the disc or movie file. Example command-line:

```
java -jar jpsxdec.jar -x <indexfile> -i <video#> -replaceframes <xmlfile>
```

10.1 xmlfile

To use this feature, you must create a .xml file that tells jPSXdec what frames to replace, and how to replace them. The format is as follows:

```
<?xml version="1.0"?>
<str-replace version="0.2">

    <replace frame="10">newframe10.bmp</replace>

    <replace frame="#13" format="mdec">newframe13.mdec</replace>

    <partial-replace frame="@22354" tolerance="5"
                     mask="test.png" rect="20,15,200,150">
        newpartialframe.png
    </partial-replace>

</str-replace>
```

The root `<str-replace>` tag needs the `version` attribute to equal the current file format version (0.2).

Within the <str-replace> tag, only <replace> and <partial-replace> tags are allowed.

The <replace> tag specifies that a frame should be completely replaced. The frame attribute is required which specifies the frame number to replace. The frame number can be specified by the methods described in chapter 8.

The tag has one optional format attribute that can have the value of either "bs" or "mdec" which indicate that the replacement image isn't a normal image file but a 'bitstream' or 'mdec' file.

Within the <replace> tag is the file name of the image that should be encoded and inserted into the movie. Only .bmp and .png images are supported, unless the format attribute is used.

The <partial-replace> tag specifies that a frame should only be partially replaced. This is the perfect option for adding subtitles or other partial adjustments to a frame. The rest of the frame remains unchanged, so the quality loss due to re-encoding is minimized. This tag also has the required frame attribute. jPSXdec automatically compares the existing frame in the movie with the new image to detect what parts are different. It then has three optional attributes that let you specify extra pixels to ignore, even if they are different.

The tolerance attribute is similar to the 'tolerance' option for the wand selection or fill tools of advanced image editors. When detecting what pixels have changed, jPSXdec will ignore different pixels if they are only different by the tolerance amount (on each RGB channel). 0 means no tolerance: all differences are detected. 255 means all differences are ignored. The default value of this attribute is 0. jPSXdec compares the new frame against a frame that was decoded with the highest quality. It's possible some pixels might be different. In that case increase the tolerance a little.

The mask attribute lets you use an image that will indicate exactly what pixels to check for differences. Any pixel that is solid black in the mask image will be ignored. Non-black pixels will be compared according to the tolerance option.

The rect attribute is similar to the mask option. It lets you specify a simple rectangle region to compare. Pixels outside the region are ignored and considered unchanged. The value of this attribute is four numbers separated by commas: x, y, width, height. Pixels within the rectangle are compared according to the tolerance option.

Within the <partial-replace> tag is the name of the image that will partially replace the frame. Only .bmp and .png files are supported. Note that if no differences are detected between the images, nothing is changed in the original.

10.2 Encoding tips and tricks

Encoding video is tricky business. To get the absolute best quality, you probably have to be somewhat of a video format guru. For those that aren't, here is a bit of information that might help you.

Each video frame has a certain amount of space allocated to it. Simpler video frames don't

use all of the space, but the more complicated frames do.

When encoding new video frames, jPSXdec tries to encode with the best quality that will still fit in the location of the existing frame. During the replacing process, you will see messages when an encoded frame fails to fit within the allotted space. In those cases, jPSXdec will reduce the image quality a little bit and try again. It will continue to do this until the new frame image fits.

In some cases the new frame won't fit even with the worst quality, or the resulting frame is so compressed that the quality is unacceptable. These replaced frames will contain many visual artifacts. So what can you do in those cases?

PlayStation video uses the same compression technique as JPEG. The main factors that affect JPEG size are **contrast** and **color**.

Contrast Hard lines, or extreme differences in light and dark or color has the biggest impact on compression size. To reduce these, you could try smoothing the hard edges with a blur tool. To reduce the color contrast, you could try dimming the bright colors or increasing the brightness of dark colors.

Color Color doesn't affect the size anywhere near as much as contrast, but still plays a part. Any shade of gray is smaller than any kind of color. If you must have color, keeping the colors to shades of pure red or pure blue will also save you some space.

When testing your newly encoded movie, you might find the PlayStation stuttering or even freezing. You could try reducing the image contrast/color even further. Or, if you are trying partial-replace, you could try full frame replace. Unfortunately, even if you do everything right, the game may still have problems with some changed frames. It's hard to say exactly what is wrong without debugging PlayStation assembly code.

10.3 About encoder quality

The jPSXdec video encoder uses double-precision floating-point math. This should already makes it at least as good as the AVI2STR/movconv tool found in the PSX Dev kit.

There are however some advanced encoding features I wish the jPSXdec encoder could use that would make it undeniably better than the PSX Dev kit tool.

- Smart chroma sub-sampling calculation
- Pre-anti-aliasing and pre-blockiness reduction
- Temporal spreading of quantization error
- Shorter VLC adjustment searching

If there are any insights you could provide about these very awesome enhancements, please let me know.

11 Other notes

Distributing game patches: jPSXdec’s ability to replace game content is awesome, but was

never designed to be included in a patch distribution. I recommend you use other ways to distribute a game patch, such as xdelta.

The AVI files jPSXdec generates are not designed to be a good format for wide distribution. They should be edited as needed, then re-encoded in a good format for distribution, such as mp4.

11.1 jPSXdec compatibility with other media programs

If it was difficult to get the PlayStation colors correct for 10 years before jPSXdec, it's an additional challenge getting any colors to work right with other multimedia programs. Over the years I've made comparisons between several different programs, testing to see which is better, worse, or different. There are three issues to consider, each could deserve its own chapter:

- YUV color range: either Rec.601 described in section 7.2.5 vs. pc.601 as described in section 7.2.6
- Chroma placement: the PlayStation uses MPEG1 style placement
- Variable frame rate

Over the years I've observed how different programs handle these three factors. But please double-check these yourself. Things may have changed since writing this.

VirtualDub – treats MJPG differently than JPG. For MJPG it uses the limited Rec.601 YUV, but for JPG it uses the broader pc.601 JYUV. So if you open a MJPG video generated by jPSXdec in VirtualDub, the colors will be different.

Windows Media Player – Uncompressed RGB videos looks interlaced for some reason. MJPG shows a green glow on the edges. Also doesn't play AVI files with variable frame rates correctly.

Windows "Movies & TV" – Doesn't play AVI files with variable frame rates correctly.

ffmepg (and ffmpeg based programs, which is nearly all open source media programs, such as VLC) – Saw no obvious issues using these with jPSXdec generated image/video files, though it would be worth checking the chroma shift.

AviSynth – AviSynth has the most flexible and powerful audio/video processing power of anything I've seen. You can define *explicitly* how videos should be interpreted. It is the only program that can properly read the pc.601 JYUV AVI files described in section 7.2.6 when using these settings:

```
AviSource("jPSXdec-JYUV.avi", pixel_type="YV12")
ConvertToRGB32(matrix="pc.601", ChromaInPlacement="MPEG1")
```

K-Lite codecs, commercial programs, and dozens more... – Unfortunately the specifications of AVI and MJPG formats are vague, so every program could do something differently. When in doubt, use jPSXdec to generate a .png image sequence, and compare that to what another program is doing.

11.2 A final note about the meaning of “quality”

The term “quality” is used very often in this manual, but quality is difficult to define, and is often subjective. What kind of quality are you looking for?

- How closely can we recreate the game experience?
  - The video decoding process used by the PlayStation hardware is ultimately written to the video memory. We have come close to reproducing the exact bytes written to the video memory, but still not perfectly. I have run across a person on the internet who was working to reverse engineer the actual PlayStation hardware, including the MDEC chip used to decode video. If that work could be recreated in code, maybe we’d get a perfect decoding process.
  - However, that would only give us the bytes written to the video memory. From there it is sent as a video signal to the television. That process also modifies the final image displayed on the pixels on the screen. At the time, those were CRT screens, but now we have LED and others. We’re also entering an age of HDR. Which of these are the quality you are looking for?
- But what about the original media used to create the video files we find on PlayStation discs? The source material is definitely better than the end product. How close can we get to recreating that original quality? Is that what you are looking for?

12 Reporting issues

Feel free to email me using the address at the top of this file, or report the issue on the project website. Ideally, please include the index (.idx) of the file you're having issues with, along with the generated debug00.log file after reproducing the issue.

13 Building from source

jPSXdec can be built using the readily available Apache Ant build system. An Ant build script has been included. If Ant is installed and configured properly, simply running the `ant` command in the directory with build.xml will generate a directory containing the binary program and everything needed to distribute it. LibreOffice will be needed to convert this manual to .pdf format.

14 Translation

The Spanish translation of jPSXdec has been made possible by the generous contributions of Víctor González and Sergi Medina. The Italian translation was generously contributed by Gianluigi "Infrid" Cusimano. The Japanese translation was generated using Google translate.

If you would like to translate jPSXdec into your own language, please reach out to me using the email at the top of this manual, or on the project site.

15 License

jPSXdec is licensed under a non-commercial license. See LICENSE.txt for details. If you would like to use jPSXdec in a way that is incompatible with this license, let me know. I am quite willing to make exceptions on a case-by-case basis. If you think your request is reasonable, I probably will too.

16 Disclaimer