Indiana University
Media Digitization and Preservation Initiative (MDPI)

White Paper:
Encoding and Wrapper Decisions and Implementation for
Video Preservation Master Files

Authored by:

Mike Casey, Director of Technical Operations, MDPI

Reviewed by:

Carla Arton, Film Digitization Specialist, IU Libraries Moving Image Archive
Jon Cameron, Digital Media Service Manager, IU Libraries
Jon Dunn, Assistant Dean for Library Technologies, IU Libraries
Heidi Kelly, Digital Preservation Librarian, IU Libraries
Brent Moberly (Appendix author), Software Developer, UITS, IU
Brian Wheeler, Senior Systems Engineer, IU Libraries

Special thanks to Dave Rice for astute comments and suggestions

Release date: March 27, 2017

Indiana University
Media Digitization and Preservation Initiative

Encoding and Wrapper Decisions and Implementation for
Video Preservation Master Files

1. Overview

There is no consensus in the media preservation community on best practice for encoding and wrapping video preservation master files. Institutions engaged in long-term video preservation typically choose from three paths, each of which are currently seen as viable for this purpose:

- 10-bit, uncompressed, v210 codec, usually with a QuickTime wrapper
- JPEG 2000, mathematically lossless profile, usually with an MXF wrapper
- FFV1, a mathematically lossless format, with an AVI or Matroska wrapper

No single path can be said to be widely adopted, in part because there are relatively few institutions engaged in digitizing video for long-term preservation, especially at scale. It appears as of this writing that institutions are roughly evenly divided between the three paths listed above.

The intense interest and uncertainly in the media preservation community surrounding these issues prompted us to write this document. In doing so, our objective is to report in detail on a set of choices and an implementation that has worked well for our preservation project. We do not hold this up as a unitary best practice or as a solution for every project. Rather, we hope that it will provide useful information to others who are walking the same path.

2. IU Application

Indiana University engaged with this issue for its Media Digitization and Preservation Initiative (MDPI)[1] which is charged with digitally preserving all significant audio and video recordings on all IU campuses in time for the University Bicentennial in 2020. Under this initiative, IU expects to digitize approximately 78,000 standard definition video recordings. The bulk of the digitization work is undertaken by IU's private partner, Memnon Archiving Services, with the remainder completed by a smaller IU-run facility. Memnon mainly uses parallel transfer workflows where one operator digitizes more than one videotape at a time. IU Media Digitization Studios handles tapes with problems

---

[1] See https://mdpi.iu.edu/ for further information on MDPI

that are not appropriate for a parallel transfer workflow and digitizes them primarily with a 1:1 workflow.

3. History of this issue at IU

3.1 Planning project working group

Indiana University's earlier Media Preservation Initiative (MPI) planning project engaged in an extensive review of video preservation options in 2011-12. At the time, path 3 above (FFV1) was not widely known and was adopted by only a couple of institutions. MPI chose the 10-bit uncompressed path, in part because of interoperability problems with different implementations of JPEG 2000/MXF lossless as well as greater technical complexity and a lack of available tools for this option. It was felt that JPEG 2000/MXF presented greater risk for us over time. The uncompressed option was viewed as simpler, more widely adopted at the time particularly within the archival community, and presenting fewer barriers to long-term preservation and access. This decision was affirmed again later in the MPI planning project.

MPI chose 50 mbps MPEG-2 I-frame only as the codec for generating a single high-quality "production master" (often called a "mezzanine" file). This format and its particular configuration is in wide use in the broadcast community and is well-supported. Restricting 50 mbps MPEG-2 to "I-frame only" preserves the integrity of the frames from the analog source and enables easier editing.

3.2 Reconsideration under MDPI

In 2015 it appeared that FFV1 was seeing greater adoption and emerging as a compelling alternative to uncompressed and JPEG 2000. The IU Media Digitization and Preservation Initiative (MDPI) decided to reopen this decision and initiated a research and review process. This process included the following steps:

3.2.1 Exit strategy research

This work involved defining a viable exit strategy by testing transcoding of FFV1 files to uncompressed v210. Testing was carried by Brian Wheeler, Senior Systems Engineer for IU Libraries.

3.2.2 Capture research

Memnon would use FFmpeg within their custom-built encoding system to generate FFV1 files. However, the IU side of the operation also would require a way to create FFV1 files. This work involved identifying tools that could be used for this purpose.

3.2.3 Comparison of issues

Utilizing work published by the Federal Agencies Digitization Guidelines Initiative (FADGI),[2] MDPI constructed a comprehensive comparison spreadsheet of the various video preservation digitization alternatives. This spreadsheet was edited from the FADGI original to include only alternatives and issues relevant to MDPI's interests. This facilitated direct comparison of the alternatives, identification of potential risks, and assessment of advantages and disadvantages.

3.2.4 Consultation with an outside expert

MDPI engaged Chris Lacinak from AVPreserve for a phone call in which he and the IU team discussed the advantages and disadvantages of the various alternatives.

3.3 Summary of research results

In our research into exit strategies we were able to move FFV1 files to a lossless codec (H.264 lossless, JPEG 2000 lossless, VP9 lossless) with no loss of data. We were also able to move to v210 uncompressed with minor and probably irrelevant loss. Working with Dave Rice, we discovered that v210 itself is not strictly lossless. If implemented correctly, v210 reserves the very high and very low sample values for synchronization purposes. Observed samples in the reserved ranges are set to the closest valid value. Not all encoders follow the v210 specification correctly – for example, Blackmagic will use the reserved values for accurate sample values, thereby providing a true lossless uncompressed file that is, nevertheless, not fully compliant with the specification. We further roundtripped a valid v210 file through FFV1, resulting in exactly the original content at the end of the process.

For further protection we discussed escrowing a copy of FFmpeg source (and/or a working VM image with FFmpeg on it) if we are concerned that the format will not be supported in the distant future.

Research into capture tools revealed that there are very few for FFV1. Our options appeared to be the following:

- vrecord – a new application that works on a Mac
- Virtualdub with ffdshow-tryouts—Windows, 8 bit only, FFV1 version 1.1 only
- capture to v210 first then transcode to FFV1 using FFmpeg
- capture using FFmpeg which requires developing a simple capture tool

---

[2] See http://www.digitizationguidelines.gov/guidelines/video_reformatting_compare.html

Our PC environment, the desire to use the latest version of FFV1 (version 3) for critical features, our requirement for 10 bit encoding, and our reluctance to expand and complicate the workflow by capturing to v210 first, left us with the last alternative. We developed specifications for a minimal capture interface that made use of FFmpeg for encoding and wrapping the video data, providing the IU side of the facility with what was required to capture video in this format. This also aligned us with Memnon which was using a similar approach. We agreed with Memnon to use the same FFmpeg binary so that problems on both sides of the facility are restricted to one version/approach and can be troubleshot together. While this work required some developer resources, neither IU nor Memnon found it to be a lengthy, burdensome, or particularly difficult task.

4. Technical and strategic analysis

4.1 File format

Direct comparison of the strengths and weaknesses of the various alternatives identified a number of key advantages to FFV1. Specifically,

- roughly 65% less data than a comparable file using the v210 codec
- open source, non-proprietary, and hardware independent
- largely designed for the requirements of digital preservation
- employs CRCs for each frame allowing any corruption to be associated with a much smaller digital area than the entire file
- supports wrapper independent aspect ratio, color space, and interlacement information
- alignment with FFmpeg (from which FFV1 is a byproduct) which is a well-established, widely used, and open source cross-platform solution to record, convert and stream audio and video[3]
- only two source code bases for creating FFV1 files—FFmpeg and LibAV—which greatly reduces the opportunity for interoperability problems. LibAV forked from the FFmpeg code base in March 2011
- intensive current development through the PREFORMA project in Europe to create tools and take steps towards standardization
- while none of the alternatives can be considered widely adopted, FFV1 appears to be trending upwards among developers and cultural heritage organizations engaged in preservation work

Although MDPI had developed specifications for storage and networking around 10 bit uncompressed preservation master files, it was considered a definitive advantage to

---

[3] See https://ffmpeg.org/

reduce the amount of data that the project would generate by some 65%. This would not only lessen storage costs but, of course, make network transmission consume less time, creation and validation of checksums consume less time, etc. FFV1 uses variable bit rate encoding so the size of the resulting file varies according to the nature of the program content. In our experience we have seen an average of 33.2 GB per hour of content in the FFV1 preservation master file. One blind but not entirely random sampling of 20 files showed a range stretching from 12.97 GB per hour at the low end to 48.40 GB per hour at the high-end. The uncompressed v210 codec delivers file sizes on the order of 100 GB per hour of content.

Working with open source software mitigates a number of risks associated with proprietary applications within the context of long-term, stable, sustainable preservation. This includes the risk that a vendor will go out of business or choose to stop supporting proprietary software. Open source applications permit direct access to their code base and having full rights to the source code means that a developer can make the software functional should it become obsolete. In any case, obsolescence does not appear to be an issue anytime soon as the FFmpeg and LibAV libraries are used by most software tools that work with audiovisual files.

We also felt that the current work towards standardization was an advantage. The international standards organization Internet Engineering Task Force (IETF) has chartered a working group named "Codec Encoding for LossLess Archiving and Real-Time transmission" (CELLAR) that is tasked with standardization of both FFV1 and Matroska for use in archival environments and transmission.[4] Tools for both formats are under development by the part of the open source European Commission-funded PREFORMA project that focuses on audiovisual files.[5] These applications focus on the validation and conformance checking of files against their official specifications. These efforts seemed to us to bode well for stable future use of FFV1 and Matroska within preservation environments.

4.2 Wrapper

While the IU team felt that the decision to use FFV1 was relatively easy to make, the choice of wrapper format proved more difficult. FFV1 is sometimes used with the AVI wrapper which is an older and more limited format than the alternatives. FFV1 data is also wrapped using Matroska which is a newer, open source option. Matroska is more flexible and is designed with preservation in mind but is not yet as widely used as AVI for this purpose. The MOV format is also a possibility. Our understanding is that historically

---

[4] See https://www.ietf.org/mailman/listinfo/cellar and
http://ashleyblewer.com/img/blewer_rice_ipres_status_of_cellar.pdf
for further information.
[5] See http://www.preforma-project.eu/index.html

the combination of FFV1 with MOV has presented technical problems but these apparently have been recently resolved.

After much deliberation, consultation with several experts including Dave Rice, and testing, we chose the Matroska wrapper (mkv). Matroska is an audiovisual container or wrapper format that has been in use since 2002. Both the Matroska specification and its underlying specification for EBML are at a mature and stable stage with thorough documentation and existing validators. Matroska has recently gained native support in the Windows OS and is also the basis for Google's WebM format container. A number of media communities have adopted Matroska, which has seen extensive Internet usage, because of its features including extensible structured metadata, broad support of audiovisual encodings, subtitle management, etc. Open source software developers have built tools based on the Matroska specification for many years.[6]

It is possible for every node in a Matroska file to have an embedded CRC that provides a checksum for the rest of the node. This has been incorporated into FFmpeg so every Matroska file written by FFmpeg has this internal fixity, leaving only around 100 bytes or so not protected. With this functionality, it is possible to identify the component of the file that has changed thereby enabling part of a file to be rewritten while maintaining fixity for other parts. As with FFV1, the standardization and tool development work described above, which includes Matroska, appeared to us a positive evolution towards sustainability for long-term preservation.

4.3 The future

Our reading of the future is that as more and more archives undertake video digitization they will not accept older and limited formats (AVI) or formats developed primarily for commercial and broadcast interests (MOV). Rather, they will be looking for standards-based, open source options with features developed specifically for archival preservation. Both FFV1 and Matroska are open source and are more aligned with preservation needs than some of the other choices and we believe they will see rapidly increasing adoption and further development. The currently active PREFORMA project in Europe will begin the standardization process for both FFV1 and Matroska as well as develop open source tools specifically addressing preservation needs. We also believe that it is more fruitful, given our specific preservation requirements, to align ourselves with the FFmpeg community rather than with QuickTime developers and Apple. Finally, Matroska is simply a more flexible wrapper option than other alternatives.

---

[6] For an overview of Matroska and FFV1 as well as an update on the status of the CELLAR working group see: Status of CELLAR: Update from an IETF Working Group for Matroska and FFV1, Ashley Blewer and Dave Rice, at https://mediaarea.net/Events/PDF/2016-10-03_iPRES_Status_of_CELLAR.pdf

5 Implementation

5.1 Software development

As discussed above, both IU and Memnon chose to develop their own applications to use FFmpeg to capture data from digitization as FFV1/Matroska files. Details of the IU capture tool are presented in the appendix below. It took one IU developer roughly one month to develop the initial release for the IU application. Of course, there has been some maintenance and updating of the software since deployment.

5.2 Quality control

IU developed a quality control program to validate that the output of both the Memnon and IU digitization operations meets IU's specification for long-term preservation. This work includes checking the FFV1/Matroska preservation master files. These files are viewed using the VLC media player, a free open source cross-platform multimedia player that supports FFV1 and Matroska.[7] Metadata relating to these files is examined using MediaInfo, a widely-used free open source program that displays technical information about media files. IU QC also uses the free open source application QCTools which, again, supports FFV1 and Matroska. QCTools enables deep, detailed inspection of video signal characteristics in order to detect error. MDPI QC workstations have been set up for 1 Gbps connections but have recently been upgraded for 10Gb over 50 micron multimode fiber optic cable which enables relatively easy downloading of preservation master files.

5.3 Unit access to files

IU media holding units—libraries, archives, departments, centers, etc.—are provided with a tool that allows them to download files for the content they steward, including FFV1/Matroska video preservation master files. This is done routinely for unit QC, researcher requests, or other access needs.

6 Conclusion

At this writing, more than 38,000 video files have been created using FFV1 and Matroska at Indiana University. Most of these were created by Memnon with a smaller number recorded from digitization of problem tapes by IU Media Digitization Studios. We have chosen two file formats that are open source, developed in part with preservation in mind, and on the road to standardization with tools in active

---

[7] See http://www.videolan.org/vlc/index.html

development. We have aligned ourselves with the large and active FFmpeg community rather than a private company. While the future is ultimately unknowable, we believe that this positions us well for long-term preservation of video-based content.

7 Appendix: IUMDS software implementation

IUMDS developed a capture application using FFmpeg to create FFV1/Matroska files. Below is a description of this tool. Source code is available from https://github.com/IUMDPI/IUMediaHelperApps

7.1 Overview

The utility is written in C#. It "drives" FFmpeg, which does all of the actual recording. The recorder has additional functionality to support barcode scanners and to provide rudimentary audio monitoring for the engineer.

The barcode scanner functionality allows the engineer to scan barcodes on objects to be preserved rather than having to type them in manually. Code was adapted from these two sources to implement the recorder's barcode functionality:

- https://github.com/aelij/RawInputProcessor
- http://www.codeproject.com/Articles/17123/Using-Raw-Input-from-C-to-handle-multiple-keyboard

For audio monitoring, the recorder uses the NAudio library (https://github.com/naudio/NAudio). Essentially, the audio monitor allows the engineer to verify that sound is coming into the system from the recording devices. Finally, the utility uses FFPROBE to inspect the files it produces.

In terms of operation, there are three general steps: create, record, and combine.

7.2 The create step

In the "create" step, the engineer specifies the barcode, a file sequence number, the file use, and the number of channels to record. The utility uses these selections to generate a file name to use when recording and to drive FFmpeg in the recording step.

Here is a screenshot of the create panel:

7.3 The record step

The "record" step controls FFmpeg while providing a degree of status feedback to the engineer. Here, the engineer can pause and restart the recording as required. The record panel includes two duration timers, which display the total recording time and the duration of the part in question, if the engineer has paused and resumed recording:
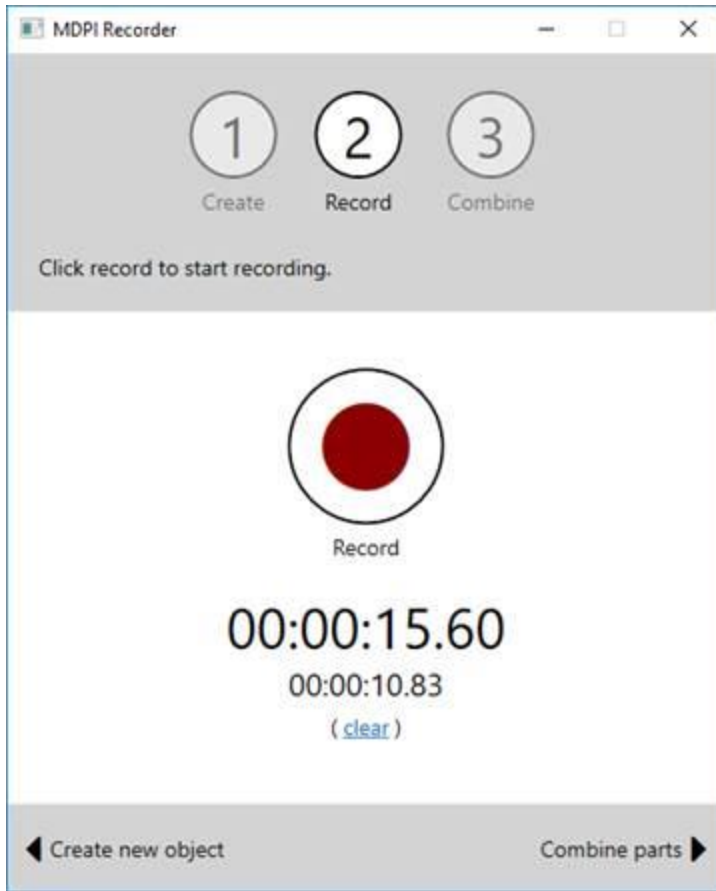
In the example, the utility has recorded 15.60 seconds of total video, with the current part containing 10.83 seconds. While FFmpeg is running, the utility uses data from the current FFmpeg log to advance both times. After recording has finished, the utility uses FFPROBE to calculate the total duration of all parts more accurately.

The "clear" option allows the engineer to start over---it deletes the temporary recording files and resets both timers.

The "pause" functionality was a bit of a challenge to add, as FFmpeg has no concept of "pause." Basically, the utility stops and restarts FFmpeg when the engineer hits pause. Pausing and restarting the recording produces independent parts files which are then concatenated in the "combine" step to make one complete file.

These parts are stand-alone video files in their own right, meaning they can be opened and viewed, etc. The "parts" functionality is in place to allow the engineer to pause and resume recording.

Part files are stored in the utility's temporary directory with filenames indicating their order. This is what they look like on the file system:
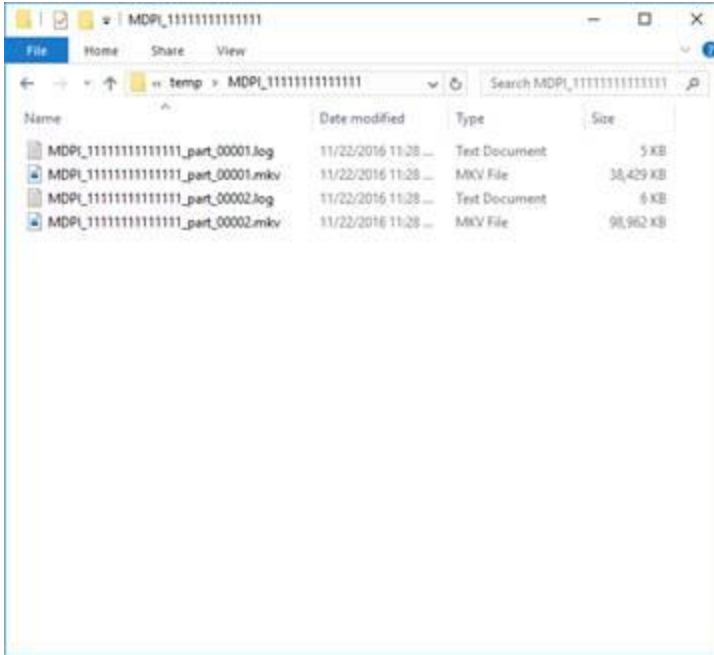
The utility also captures logs for each part it produces. These logs are stored alongside the part files in its temporary folder.

While recording, the utility streams the contents of the current log file to its output window. This window contains a volume display, a frame counter, and the text of the current log:
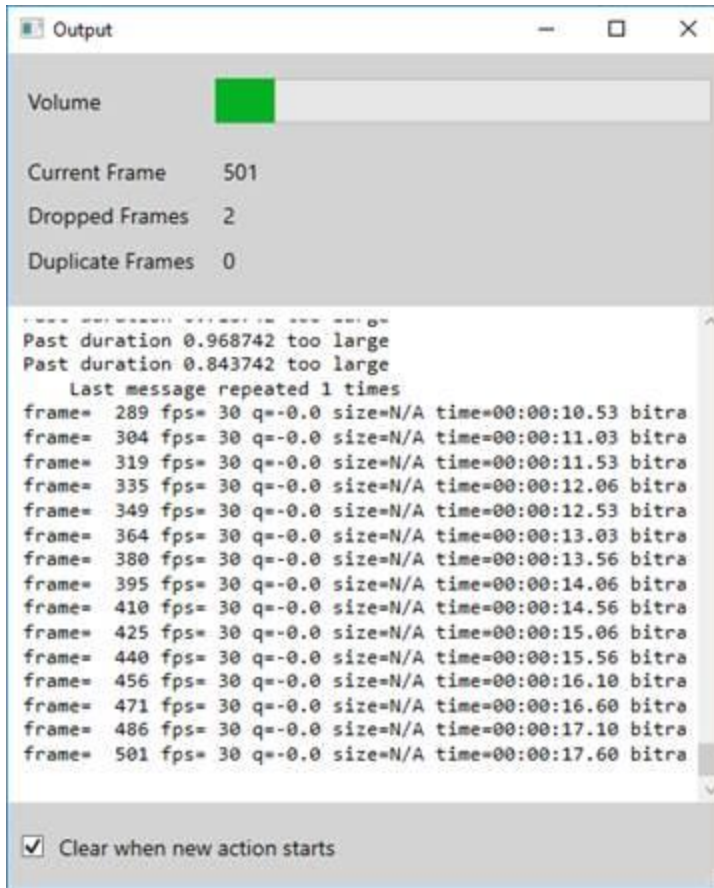
The volume meter is rudimentary at best. Its purpose is not to replicate the functionality of the volume meters on the engineer's capture equipment, but only to provide visual confirmation that sound is being captured as well as video.

The frame counter tracks the current frame and provides dropped and duplicate frame counters. This information comes from the recording log and alerts the engineer if there is a large number of dropped or duplicate frames.

The recording itself is done by FFmpeg. The utility generates a command-line based on the options the engineer selected in the "create" step and starts FFmpeg with this command-line.

This is the base production command-line:

-rtbufsize 1500M -stats -f decklink -i "DeckLink Studio 4K@1" -acodec pcm_s24le -strict -2 -ar 48000 -vcodec ffv1 -level 3 -threads 8 -coder 1 -context 1 -g 1 -slices 24 -slicecrc 1 -pix_fmt yuv422p10le

In two-channel mode, the recorder simply appends the appropriate file name to this command-line. For example:

-rtbufsize 1500M -stats -f decklink -i "DeckLink Studio 4K@1" -acodec pcm_s24le -strict -2 -ar 48000 -vcodec ffv1 -level 3 -threads 8 -coder 1 -context 1 -g 1 -slices 24 -slicecrc 1 -pix_fmt yuv422p10le MDPI_11111111111111_part_00001.mkv

In four-channel mode, the recorder applies FFmpeg's "channel split" filter as well:

-rtbufsize 1500M -stats -f decklink -i "DeckLink Studio 4K@1" -acodec pcm_s24le -strict -2 -ar 48000 -vcodec ffv1 -level 3 -threads 8 -coder 1 -context 1 -g 1 -slices 24 -slicecrc 1 -pix_fmt yuv422p10le -filter_complex channelsplit=channel_layout=15 MDPI_11111111111111_part_00001.mkv

The two-channel command-line produces a video file with a single two-channel audio stream, while the four-channel command-line produces a video file with four single-channel audio streams (one for each channel coming in from the engineer's capture devices). Note that filenames include a part specifier ("_part_00001").

7.4 The combine step

The "combine" step creates the final video file. If there is only one part, the utility simply copies that part to the output folder with the correct filename. If there is more than one part, the utility uses FFmpeg to concatenate the parts into the final file. To do this, it generates a directive file ("combine.txt") containing all of the target parts, and then runs FFmpeg with the following command-line:

-y -f concat -i "combine.txt" -c copy -map 0 "MDPI_11111111111111_01_pres.mkv"

This produces a single file from each of temporary parts file. We do not need to specify format details here because all of the parts generated in the record step have the same audio and video format, so FFmpeg can just combine the files without additional conversion.

7.5 Configuration

The recorder is configured via its xml app.config file, Recorder.exe.config. This file allows for the following to be specified:

ProjectCode	The project code to use when generating filenames
PathToFFmpeg	The path to FFmpeg
PathToFFPROBE	The path to FFPROBE
OutputDirectoryName	The path to the folder that will receive final, output files

WorkingDirectoryName	The path to the utility's working (or temp) folder. This is where the utility saves part files and logs.
AudioDeviceToMonitor	The name of the audio device to monitor while recording. This will vary with your hardware (see below)
FFmpegArguments	The base arguments to use when running FFmpeg
BarcodeScannerIdentifiers	A comma-separated list of identifiers associated with barcode scanners attached to the system

In the case of the "AudioDeviceToMonitor", the utility will suggest valid values if none are present.

7.6 Memnon implementation

After analyzing commercially available solutions and deciding that more flexibility was required, Memnon developed its own ingest solution. It is built with the best available hardware using primarily FFmpeg among other software applications. They report that once IU made the decision to use FFV1 with the Matroska wrapper and clearly specified metadata requirements, very little had to be adjusted to have the Memnon solution meet IU production requirements and fully integrate it into the Memnon Production Asset Management (PAM) system.