Conch – Conformance checking for media files

Project Acronym: PREFORMA

Grant Agreement number: 619568

Project Title: PREservation FORMATs for culture information/e-archives

Prepared by: MediaArea.net SARL

- Jérôme Martinez
- Dave Rice
- Tessa Fallon
- Ashley Blewer
- Erik Piil
- Guillaume Roques

Prepared for: PREFORMA Consortium

Date: December 31, 2014 Revised Date: March 2, 2015

Summary: This report presents MediaArea's research, planning, and design work to develop a shell and conformance checker tentatively entitled "Conch" with a particular focus on Matroska, FFV1, and LPCM.

  - Advance Improvement of Standard Specification
- Sustainable Open Source Business Ecosystem
- Participation at Open Source conferences
- Project Management Strategy
  - Goal:
  - Method:
  - Justification/Purpose:
  - Intended Result:
  - Risk Analysis Model
  - Internal Risk Assessment
- Introduction to Architecture
  - Applicability
  - Portability
  - Scalability
  - Distribution
  - Modularity
  - Deployment
  - Interoperability
- Global Architecture
- Checker Architecture
- Architecture schema
- Common to all elements
  - File access
  - File processing
  - Internet Access
  - Automation
  - Batching
  - Prioritization
- #
- Core (Controller)
- Database
- Scheduler
- Files listener
- implementation checker and Metadata Grabbing Module
- Policy checker
- Reporter
- User interface
- Transport layer
  - Conch: File on disk or direct memory mapping
  - Plugin integration proof of concept: libcURL
- Container/Wrapper implementation checker
  - Conch: Matroska checker
  - Plugin integration proof of concept: mkvalidator
- Container/Wrapper Demultiplexing
  - Conch
  - Plugin integration proof of concept: FFmpeg
- Stream/Essence implementation checker
  - Conch:
  - Plugin integration proof of concept: jpylyzer

- Contribution Guide
  - File Naming Conventions
  - Rules for Qt/C++ code
  - Guidelines for C++ code is as follows:
  - Rules for contributing code
  - Rules for contributing feedback
  - Linking
  - Test Files
- Release Schedule
- License
- Conclusion

Project Introduction

The PREFORMA challenge illuminates and responds to a significant and real obstacle that faces the preservation community today. This report details MediaArea's design plan to create a toolset (tentatively entitled "Conch") consisting of an implementation checker, policy checker, reporter, and fixer of a select list of formats.

As preservation workflows have incorporated digital technology, significant amounts of careful research have gone into the selection of file format recommendations, lists of codec specifications, and development of best practices; however, despite the existence of such recommendations, there remains a lack of assessment tools to verify and validate the implementation of such recommendations. A few validation tools (such as mkvalidator) are produced alongside the development of their associated standards; however, most file format specifications are not officially tied to any validation tool and are documented through human-readable narrative without equivalent computer-actionable code. Where a metadata standard may be described in both a data dictionary and a computer-usable XML Schema, file formats standards often lack a computer-usable verification method. The PREFORMA project recognizes this discrepancy and the resulting long-term impacts on archival communities and seeks to fill in the gaps necessary to provide memory institutions with levels of control to verify, validate, assess and repair digital collections.

MediaArea's approach to this challenge centers on FOSS (Free and Open Source Software), modular design, and interoperability and will rely strongly on MediaInfo (an open source MediaArea product) to meet this challenge. MediaInfo is often advised as the first tool to use when a media file is not playable, allowing the user to identify characteristics that would help find an appropriate playback or transcoding tools. MediaInfo's open licensing and agility in technical metadata reporting have encouraged its integration into several archival repository systems and OAIS-complaint workflows to assist archival with technical control of collections.

MediaArea sees community involvement as a key factor of evaluating the success of the project. To encourage this, during the prototype phase MediaArea will perform the development work for command line utilities, graphical user interfaces, and documentation in publicly accessible repositories at github.com. Mediaarea will also set up an online set of project resources such as public access to a corpus of test media, an IRC channel, and a responsive public issue tracker.

In order to foster and demonstrate a focus on interoperability throughout the project, MediaArea will work with Artefactual in order to facilitate integration of resulting project components into Archivematica, a digital repository focused on OAIS. This collaboration will bring the availability of additional OAIS and digital preservation expertise to the project and provide an additional means for the project deliverables to be made available to users.

Introduction of Featured Formats

During the development phases MediaArea will focus on one container format, Matroska, and two streams, LPCM and FFV1. The design work of MediaArea will address formats and codecs through a modular architecture so that other formats or codecs may easily be added alongside or after development.

Matroska, FFV1, and LPCM describe very unique concepts of information including:

- a container format, Matroska
- an audio stream, LPCM
- a video stream, FFV1

These three information concepts will inform distinct user interface and reporting design in order to process these concepts through differing strategies. For instance, reporting on the status of 100,000 of video frames within a video recording may be done more efficiently in a different interface as one designed to communicate the hierarchical structure of a file format.

Additionally other formats currently being addressed by PREFORMA fit within these three conceptual categories; for instance, PDF and TIFF are formats (containers) and JPEG2000 is a video stream. These three concepts affect the design of an overall application shell as conformance information for each category can have its own optimized user interface.

Matroska

Matroska is a open-licensed audiovisual container format with extensive and flexible features and an active user community. The format is supported by a set of core utilities for manipulating and assessing Matroska files, such as mkvtoolnix and mkvalidator.

Matroska is based on EBML, Extensible Binary Meta Language. An EBML file is comprised of one of many defined "Elements". Each element is comprised of an identifier, a value that notes the size of the element's data payload, and the data payload itself. The data payload can either be stored data or more nested elements. The top level elements of Matroska focus on track definition, chapters, attachment management, metadata tags, and encapsulation of audiovisual data. The Matroska element is analogous to QuickTime's atom and AVI's chunk.

Matroska integrates a flexible and semantically comprehensive hierarchical metadata structure as well as digital preservation features such as the ability to provide CRC checksums internally per selected elements. Because of its ability to use internal, regional Cyclic Redundancy Check (CRC) protection it is possible to update a Matroska file during OAIS events without any compromise to the fixity of its audiovisual payload.

Matroska has well written documentation and a draft specification but is not defined through an external standards organization although some drafts for such work have already been produced.

FFV1

FFV1 is an efficient lossless video stream that is designed in a manner responsive to the requirements of digital preservation. Version 3 of this lossless codec is highly self-descriptive and stores its own information regarding field dominance, aspect ratio, and colorspace so that it is not reliant on a container format to store this information (other streams that rely heavily on its container for technical description often face interoperability challenges).

FFV1 version 3 mandates storage of CRCs in frame headers to allow verification of the encoded data and stores error status messages. FFV1 version 3 is also a very flexible codec allowing adjustments to the encoding process based on different priorities such as size efficiency, data resilience, or encoding speed.

The specification documentation for FFV1 is partially complete and has recently been funded by vendors utilizing FFV1 as a codec for audiovisual preservation and large-scale digitization efforts.

Linear PCM

Linear Pulse Code Modulation (LPCM) is a ubiquitous and simple audio stream. PCM audio streams may be comprised of signed or unsigned samples, arranged in little-endian or big-endian arrangements, in any number of audio channels. PCM is very flexible but is not self-descriptive. A raw PCM file can not properly be decoded without knowing the sample encoding, channel count, endianness, bit depth, and sample rate. LPCM is typically dependent on its container format (such as WAV) to store sufficient metadata for its decoding.

Because PCM streams contain only audio samples without any codec structure or metadata within the stream, any data by itself could be considered valid PCM and decoded as audio. Determining the conformity or technical health of PCM data requires the context of information provided by its container format.

Development of a conformance checker

MediaArea's design of the conformance checker is intended to allow interoperability between the conformance checkers of PREFORMA's other suppliers so that users may integrate multiple conformance checkers within a single 'shell'. The PREFORMA project is comprised of four components:

- implementation checker
- policy checker
- reporter
- metadata fixer

The PREFORMA project must document and associate implementation or policy rules with data types (such as formats, streams, frames, etc) and authorities (such as specifications, community practices, or the local rules of a memory institution). MediaArea recommends that communication between the implementation checker and the shell be performed through an API designed via collaboration of the PREFORMA suppliers.

Implementation Checker

For each supported format (Matroska, FFV1, and LPCM), the implementation checker should assess compliance and/or deviation between files and a series of adherence checks which are written by dissecting rules and logic from each format's underlying specifications, including rules that may be deduced or inferred from a close reading of the specification or related authoritative documentation. MediaArea has drafted registries of conformance rules within the PREFORMA design phase and plans to collaborate with each format's specification communities to refine them. See the Conformance Check Registry.

For streams such as FFV1, implementation checks may be performed frame-by-frame to discover frame-specific issues such as CRC mismatches, invalid frame headers, or incomplete frames. Frame-by-frame conformance assessments will naturally be time consuming as nearly the entire file must be read. In order to accommodate user's various time priorities the checker will use options to perform checks on the first few frames of a stream, a percentage of the frames, or all of the frames.

MediaArea has drafted a registry of metadata elements to be used in described an implementation check, which provides unique identifier, the scope, and underlying rationale and authority for the check. Code created to preform checks will be internally documented with references to conformance check's unique identifiers, so that MediaArea may create resources for each conformance check that relate the identity of the check, its underlying authority, sample files, and associated code.

Policy Checker

For each format addressed through a implementation checker, MediaArea will create a vocabulary list of technical metadata and contextual descriptions. Additionally, MediaArea will define a list of operators to enable various comparators between the actual technical metadata and any expected user-provided metadata. Such defined language will allow users to make policy check expressions such as:

- FFV1.gop_size MUST_EQUAL "1"
- FFV1.slice_crc MUST_BE_ENABLED
- FFV1.version GREATER_THAN_OR_EQUAL "3"
- MKV.tag.BARCODE MUST_START_WITH "ABC"
- MKV.tag.DATE_DIGITZED IS_BEFORE "2014-01-01"
- MKV.tag.ISBN MATCHES_REGEX "$(?=[\text{-0-9xX }]\{13\})(? : [0-9] + [-])3[0-9] * [xX0-9]$"

MediaArea proposes that PREFORMA suppliers collaborate to define a common expression for sets of policy checks via an XML Schema, associated data dictionary, and vocabulary of comparative operators. The collaboration would include agreement and definition on the operators ("Greater Than", "Starts With", etc) of the policy checks and attempts to normalize technical metadata between common formats where they have overlapping concepts. Each policy checker would produce a vocabulary of technical metadata specific to its format for policies to be checked against as well as inclusion within an API so that the shell can access the possible operators of any enabled implementation checker.

MediaArea will provide sample sets of policy checks based on interviews with memory institutions and community practice.

Reporter

MediaArea proposes that PREFORMA suppliers collaborate to create a common XML Schema to define the expression of PREFORMA reporting (referred herein as "PREFORMAXML"). The schema should define methods to express technical metadata and relates checks to formats/streams (including components of formats and streams such as frames or attachments). The XML Schema should encompass not only information about the files being assessed but also the conditions and context of the particular use (which shell was used, with what policy sets, at what verbosity, etc). The XML Schema should be supported by a data dictionary that is also collaboratively written by PREFORMA suppliers. MediaArea anticipates that the implementations and features performed upon the basis of a common XML Schema may vary from supplier to supplier or per implementation checker, but that adherence to a common schema is essential to interoperability and consistent user experience amongst implementation checkers.

The PREFORMAXML schema should accommodate the expression of results from multiple implementation checkers upon a single file. For example, a Matroska file that contains a JPEG2000 stream, a FFV1 stream, and a LPCM stream should be able to express one XML element about the file with sub-elements about each conformity check to reduce redundancy.

Because of Matroska's flexibility, ancillary objects like PDFs, JPEG2000 images and CRC files may also be added as attachments.

MediaArea plans to include these features commonly within MKV, FFV1, and LPCM reporters:

- Export of a standardized PREFORMAXML
- Export PREFORMAXML with gzip compression (to reduce the impact of large and highly verbose XML files)
- Export of the same data within a semantically equivalent JSON format
- Other functions based on PREFORMAXML (such as generation of PDF formats or summarization of multiple collections of PREFORMAXML) will happen within the "Shell" component

Fixer

MediaArea will produce a fixer that allows for editing the file. Enabling this function will be performed with a substantial amount of caution as in some cases a user could use it to change a file considered a preservation master. The fixer will support assessing a file first to determine the risk of editing a structurally unhealthy file and provide suitable levels of warning to the user.

The metadata fixer shall support both direct editing on the input file (with warning) or producing a new output file as a copy that the metadata change as requested.

The metadata fixer will support comprehensive logging of the change and offer options to log the performance of the edit itself with the file if it has a means to accommodate it (such as Matroska).

In addition to metadata manipulation the fixer will accommodate structural fixes to improve the structural health of the file, such as repairing Matroska Element order if ordered incorrectly, or validating or adding Matroska CRC Elements at selected levels, or fixing EBML structures of truncated Matroska files.

Substantial care should be exercised to ensure that the implementation checker properly associates risk, user warnings, and assessments with each fix allowed. In order to allow a fix the software must properly

understand and classify what may be fixed and be aware of how the result may be an improvement. Adjustments directly to a preservation file must be handled programmatically with great caution with diligent levels of information provided to the user.

An example of a fix that could be enabled in the RIFF format could be verifying that any odd-byte length chunk is properly followed by a null-filled byte. Odd-byte length chunks that do not adhere to this rule cause substantial interoperability issues with data in any chunk following the odd-byte length one (this is particularly found in 24 bit mono WAV files). If the odd-byte length chunk is not followed by a null-filled padding byte, then most typically the next chunk starts where the padding byte is and the padding byte may be inserted so that other following chunks increase their offset by one byte. This scenario can be verified by testing chunk id and size declaration of all following bytes so that the software may know beforehand if the fix (inserting the null-filled padding byte) will succeed in correcting the RIFF chunk structure's adherence to its specification.

Fixes for Matroska files could include fixing metadata tags that don't include a SimpleTag element or re-clustering frames if a cluster does not start on a keyframe.

Because many files focused on with FFV1 and Matroska implementation checkers will be quite large, MediaArea plans to provide options to either rewrite the original file with the check or edit the file in place so that the file is only changed according to the fix that is request. With the latter option is the user is 'fixing' the metadata in a 50 gigabyte Matroska file only the last few megabytes of the Matroska tagging element may be rewritten without a requirement to rewrite the non tagging elements at the beginning of the file (MediaArea deployed a similar feature within BWF MetaEdit).

Core

The shell will coordinate the actions of the implementation checker, policy checker, reporter and fixer. As PREFORMA seeks that the shell developed by each supplier supports each supplier's implementation checker(s), MediaArea encourages all suppliers to work collaboratively to negotiate API documentation to support not only our own interoperability but to support third-party development of additional implementation checkers to utilize the produced shells.

The development of the shell will strive to facilitate an intuitive and informed use by memory institutions at both expert and non-expert levels. The shell will include substantial internal documentation that mimics the online resources that we will provide so that the shell and implementation checker function well offline.

MediaArea will implement a scheduling service within the shell so that large tasks may be performed overnight or according to a defined schedule. MediaArea will enable the Shell to load queues of files from lists of filepaths or URLs. Because of the size of data involved in audiovisual checkers MediaArea will give priority to designing the shell and implementation checker to perform multi-threaded and optimized processing.

Implementation Checker (Shell) The shell produced will support all functions and requirements of the implementation checker as described as an independent utility and also support:

- Allow the user to open one or many files at a time.
- Allow the user to queue simultaneous or consecutive file analysis.
- Allow the user to select how comprehensive or verbose an conformance check may be (for instance, samples frames or all frames of video).
- Enable the user to select sections of conformance checks or sets of conformance checks that they may wish to ignore.
- Enable the user to associate certain actions or warnings with the occurence of particular checks.
- Provide feedback and status information live during the file analysis.
- (For Matroska) Present a user interface that displays the hierarchical EBML structure of the file with the corresponding policy outcome for each policy check.
- (For FFV1) Present a user interface that displays frame bitstream in a table and enable the user to filter the presentation of frame bitstream according to warnings or coherency events (for example, discontinuous aspect ratio).

Policy Checker (Shell) The shell produced will support all functions and requirements of the policy checker as described as an independent utility and also support:

- Allow PREFORMA-supported technical metadata vocabularies to be imported or synchronized against an online registry.
- Provide an interface for the user to import, create, edit, or export a valid set of policy checks.
- Implement selected set of policy checks on all open files or selected files.
- Present the outcome of policy checks in a manner that allows comparison and sorting of the policy status of many files.
- Allows particular sets of policy to be associated with particular sets of files, based on file identification or naming patterns.
- (For Matroska) Present a user interface that displays the hierarchical EBML structure of the file with the corresponding policy outcome for each policy check.
- Can display the FFV1 bitstream.

Reporter (Shell) The shell produced will support all functions and requirements of the reporter as described as an independent utility and also support:

- Export of the PREFORMAXML data at user-selected verbosity levels in a PDF format, which data visualizations supplied where helpful.
- Ability to read a collection of PREFORMA XML documents and provide a comprehensive summary and technical statistics of a collection to allow for prioritization, comparison, and planning.

Fixer (Shell) The shell produced will support all functions and requirements of the reporter as described as an independent utility and also support:

- Allow for single file or batch editing of file format metadata.
- Allow for selected metadata to be copied from one file to another or from one file to all other open files.
- Allow for file format metadata to be exported and imported to CSV or XML to enable metadata manipulation in other programs to then be imported back into the Shell and applied to the associated files.
- (For Matroska) Present a user interface that displays the hierarchical EBML structure of the file and allows the user to create, edit, or remove (with warning) any EBML element and display the associated policy or implementation check that corresponds with such actions.

Interfaces The selected formats (MKV, FFV1, and LPCM) represent substantially distinct concepts: container, video, and audio. The optimization of a implementation checker should utilize distinct interfaces to address the conformance issues of these formats, but allow the resulting information to be summarized together.

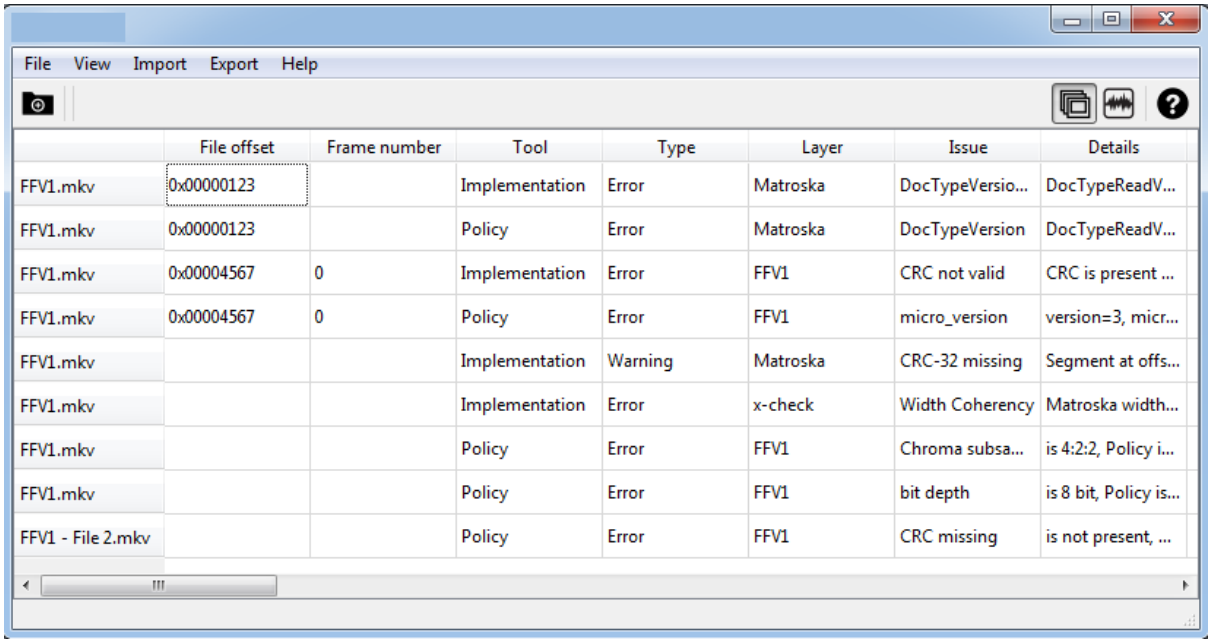
Assessment of file conformance can be displayed via a graphical user interface or a command line interface.

An interface for assessing conformance of FFV1 video could enable review of the decoded FFV1 frames (via a plugin) in association with conformance data so that inconsistencies or conformity issues may be reviewed in association of the presentation issues it may cause.

MediaArea proposes an interface to present conformity issues for audio and video streams (FFV1 and LPCM) on a timeline, so that conformance events, such as error concealment or crc validation issues may be reviewed effectively according to presentation, parent Matroska block element, or video frame.

For deep analysis, a distinct interface that allows for its hierarchical structure to be reviewed and navigated. The presentation should allow for MKV and FFV1 elements to be expanded, condensed, or filtered according to element id or associated conformity issues.

A summary of the file properties can also be displayed via a command line interface for quick reference or export.



	File offset	Frame number	Tool	Type	Layer	Issue	Details
FFV1.mkv	0x00000123		Implementation	Error	Matroska	DocTypeVersio...	DocTypeReadV...
FFV1.mkv	0x00000123		Policy	Error	Matroska	DocTypeVersion	DocTypeReadV...
FFV1.mkv	0x00004567	0	Implementation	Error	FFV1	CRC not valid	CRC is present ...
FFV1.mkv	0x00004567	0	Policy	Error	FFV1	micro_version	version=3, micr...
FFV1.mkv			Implementation	Warning	Matroska	CRC-32 missing	Segment at offs...
FFV1.mkv			Implementation	Error	x-check	Width Coherency	Matroska width...
FFV1.mkv			Policy	Error	FFV1	Chroma subsa...	is 4:2:2, Policy i...
FFV1.mkv			Policy	Error	FFV1	bit depth	is 8 bit, Policy is...
FFV1 - File 2.mkv			Policy	Error	FFV1	CRC missing	is not present, ...

Figure 1: GUI conformance output sample

```
Testing FFV1.mkv...

Implementation checking errors:
Matroska     ¦ DocTypeVersion too high    ¦ DocTypeReadVersion=4, must be lower than DocTypeVersion=2
FFV1         ¦ CRC not valid              ¦ CRC is present but is not valid, frame corrupted
x-check      ¦ Width Coherency            ¦ Matroska width=704, FFV1 width=720

Implementation checking warnings:
Matroska     ¦ CRC-32 missing             ¦ Segment at offset 0x00000123 is missing mandatory CRC-32
FFV1         ¦ micro_version              ¦ version=3, micro_version=3, this micro-version is pre-standard

Policy checking errors:
Matroska     ¦ DocTypeVersion             ¦ DocTypeReadVersion=4, Policy is DocTypeReadVersion<=3
FFV1         ¦ Chroma subsampling         ¦ is 4:2:2, Policy is 4:4:4
FFV1         ¦ bit depth                  ¦ is 8 bit, Policy is >=10
FFV1         ¦ CRC missing                ¦ is not presence, Policy is presence mandatory

Total: 3 implementation errors, 2 implementation warnings, 4 policy errors
```

Figure 2: CLI conformance output sample

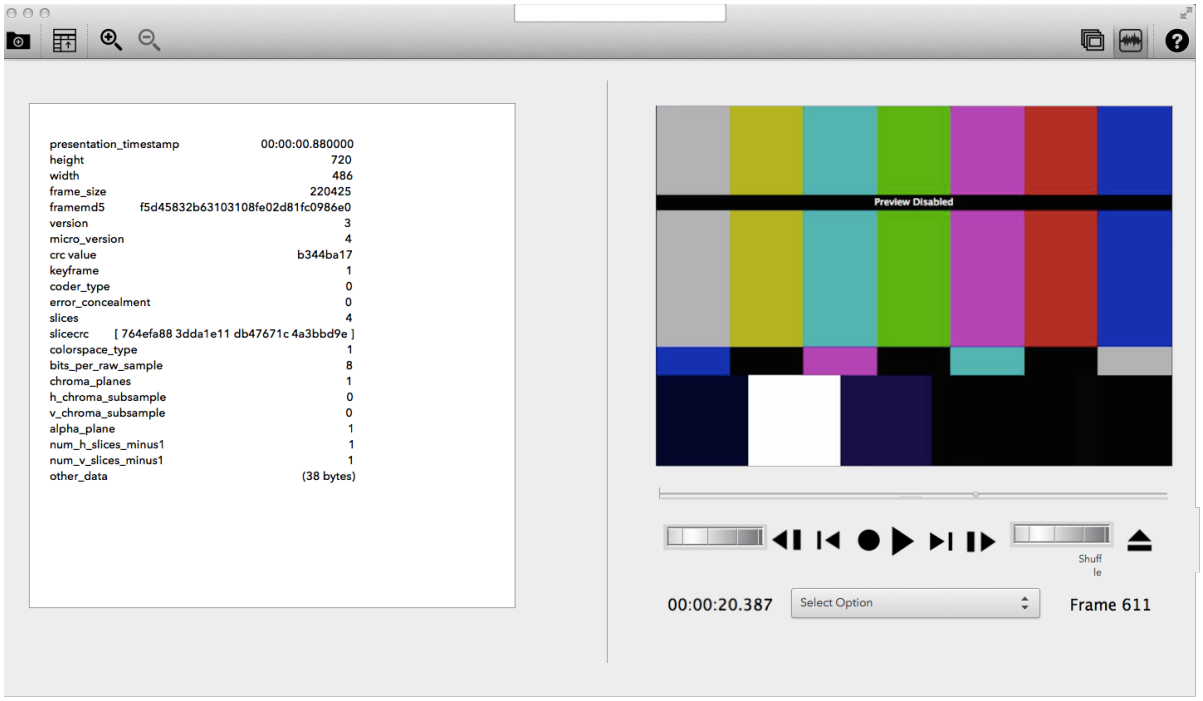


Figure 3: frame view mockup

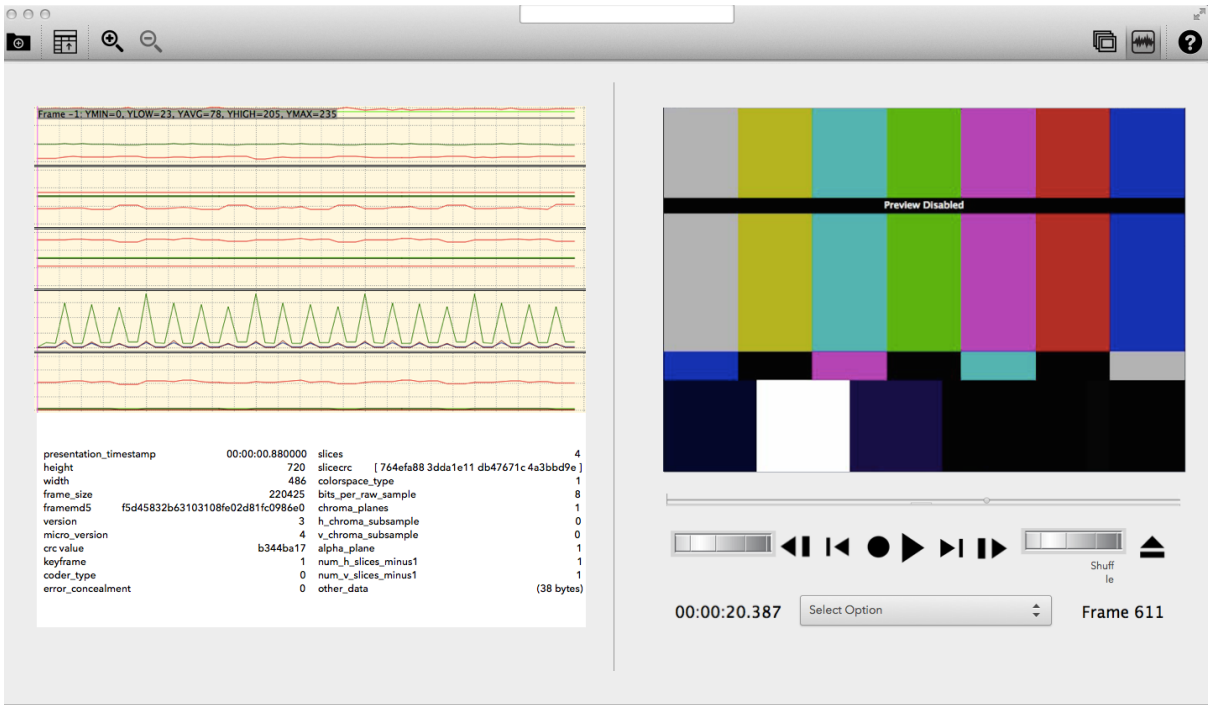


Figure 4: frame scrolling mockup

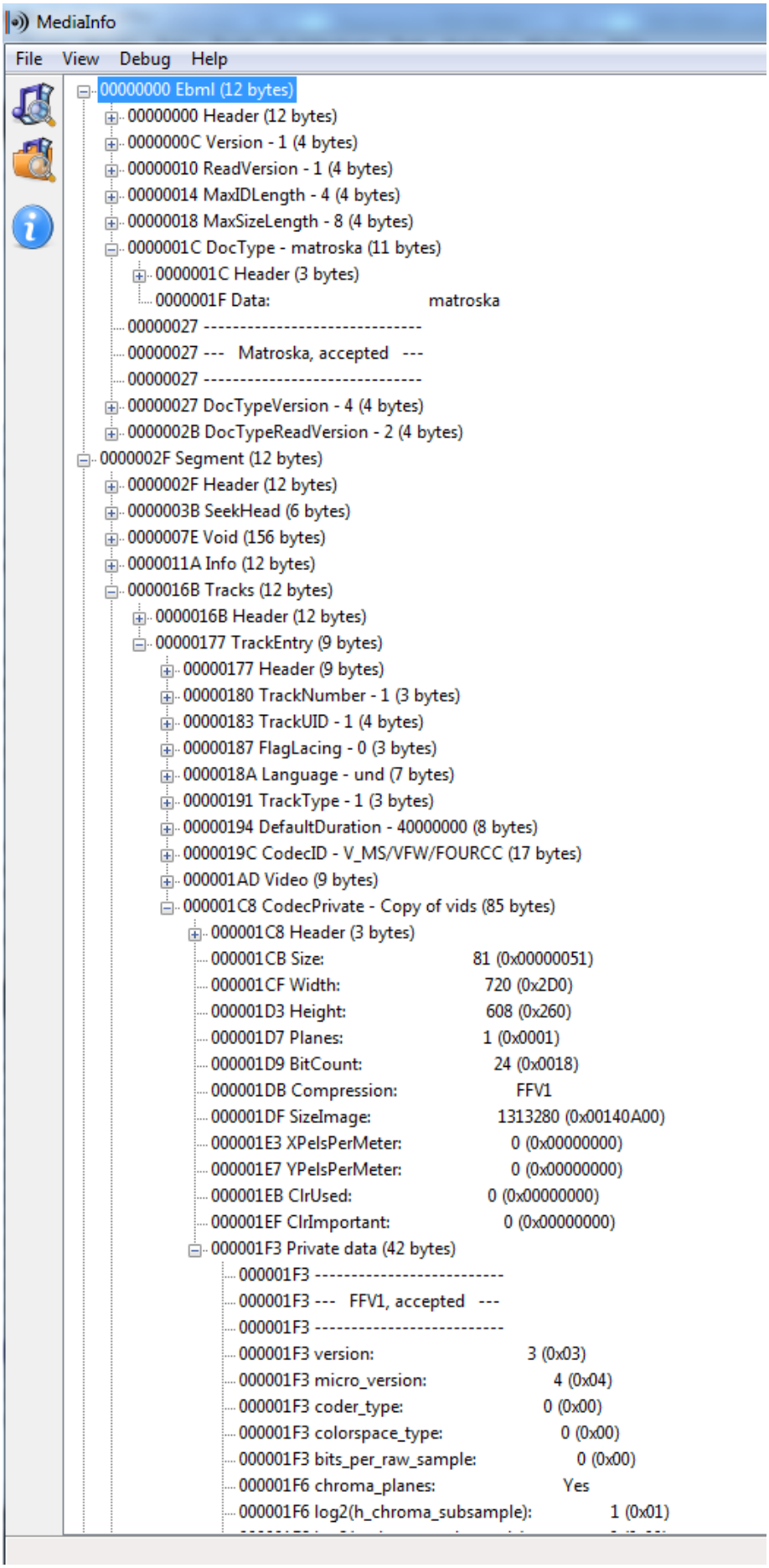


Figure 5: MediaInfo Windows

Figure 6: CLI output sample

Optimization for Large File Size

Design of a implementation checker and shell should be considerate of the large file sizes associated with video. For instance, an hour-long PAL FFV1 file (which contains 90,000 frames per hour) should provide efficient access if cases where one FFV1 frame contains a CRC validation error.

A video implementation checker should be well optimized and multi-threaded to allow for multiple simultaneous processes on video files. Additionally the implementation checker should allow a file to be reviewed even as it is being processed by the implementation checker and allow assessment of files even as they are being written.

Focus on Fixity

Both FFV1 and Matroska provide fixity features that serve the objectives of digital preservation by allow data to be independently validated without the requirement of managing an external checksumming process. FFV1 version 3 mandates CRCs on each frame. Matroska documents methods to embed checksums (MD5) in Matroska elements to allow for future validation of any content.

Although the Matroska specification states that "All level 1 elements should include a CRC-32" this is not the practice of most Matroska multiplexers. As part of the Fixer aspect of this project, MediaArea proposes to develop a implementation checker that allows users to add CRC-32 to selected elements.

The advantages of embedded fixity in preservation media files are significant. The use of traditional external checksums does not scale fairly for audiovisual files, because since the file sizes are larger than non-audiovisual files there are less checksums per byte, which creates challenges in addressing corruption. By utilizing many checksums to protect smaller amounts of data within a preservation file format, the impact of any corruption may be associated to a much smaller digital area than the entire file (as the case with most external checksum workflows).

Reference and Test Files

MediaArea anticipates creating a large corpus of reference and test files highlighting many of the issues documented in our conformance check registry. After an intital clearing of any associated rights, such files will be published under PREFORMA's selected open license agreements and disseminated for public consumption. Curated references to other relevant reference and test files in sample libraries will also be made available, which will include but not be limited to the following:

- Selections from http://samples.ffmpeg.org
- Selections from http://archive.org
- Selections from http://multimedia.cx
- PDF/A files buggy files: http://www.pdfa.org/2011/08/isartor-test-suite/
- JPEG 2000 files: https://github.com/openplanets/jpylyzer-test-files
- Matroska buggy files: Homemade + request to Matroska mailing list
- FFV1 buggy files: Homemade + request to FFmpeg mailing list
- LPCM files: Homemade

Together these references examinine a diverse set of file format expressions created by a variety of unique software platforms.

Intended Behavior by Use Case

Overview

The following use cases are presented to describe intended behaviors of the implementation checker:

Conformance Checking in an Open Archival Information System (OAIS) PREFORMA acknowledges the recommendations described in Consultative Committee for Space Data Systems' Open Archival Information System (OAIS) intended for the long term preservation of digital information (CCSDS 650.0.-B-1/ISO 14721:2003). MediaArea aims to identify all relevant areas of the OAIS reference model in relation to its proposed Conch conformance checker toolset. Moving foward, an additional examination of other OAIS-related standards will further minimize any redundancies or incompatibility with the PREFORMA project.

The OAIS reference model describes the transmission of data in the form of conceptual containers known as Information Packages. In the OAIS environment, Submission Information Packages (SIPs) are created by Producers (those who provide the information to be preserved), containing data objects with relevant packaging information. After undergoing a series of processes performed by functional entities, SIPS are transformed into Archival Information Packages (AIPs) and moved into the archive's long-term storage. Upon a query request by a Consumer, AIPs can be re-called and presented in the form of a Dissemination Information Package (DIPs) to the Designated Community.

Conformance checking services play a major role in the OAIS reference model, specifically within the functional entities during the intial transmission of the SIP. The Ingest Quality Assurance (QA) function in particular validates SIPs in the temporary storage area prior to AIP generation. Here the Conch conformance checker and its associated toolset would verify SIPs through implementation checking and policy checking with rules and specifications defined by the Archive. Following an approved check, the resulting report would be submitted as associated Preservation Description Information (PDI) within the packaging information of the AIP. However, if a conformance check rejects a SIP, administrators may choose either to consult the associated Producer, or correct related issues with the toolset's metadata fixer. (See Fig. "The OAIS Ingest Functional Entity with Conch Integration")

Policy check expressions are also useful in other functions of OAIS workflows including Archival Storage where format migration of AIPs is periodiclally undertaken. Here the comparators of technical metadata between migrated data objects is key. A conformance checker would be implemented to map all transformations through the collection of associated Preservation Description Information. For the Matroska format in particular, individual sub-element CRCs can be submitted as a check expression and later packaged with the AIP's PDI information related to Fixity. This kind of self-descriptive information can be especially useful tool for zeroing in on potentially corrupted areas of the data object.

For dissemination requests, the DIP generation fuction may include the encoding of an AIP data object to a smaller, more compressed transmission format for access. This is especially true for audiovisual data objects, whose AIPs might contain large uncompressed or mathematically lossless filetypes. Policy check expressions with the conformance checker at this phase would ensure that relationships to file characteristics like sub-sampling, bit-depth and frame rate are maintained throughout DIP generation.

Like the OAIS reference model itself, the PREFORMA conformance checker project aims to serve as a framework for standards-building activities through the creation of powerful reporting tools. By extension, MediaArea's Conch toolset enables the creation of a complex representation net in the OAIS Archive, providing information needed to adequately preserve audiovisual data objects through time.

Conformance Checking at Digitization Time

Verification of Digitization Policy Archival digitization workflows are generally highly defined and consistent so that various analog source objects are associated with particularly digitization requirements. Generally digitization specifications are selected in order to reduce alterations to the significant characteristics of the analog source material. Example of such digitization scenarios may be:

- A PAL Betacam SX tape is digitized to a Matroska/FFV1 file at PAL specifications with YUV 4:2:2 8 bit video and 4 channels of 24 bit LPCM audio
- A NTSC U-Matic tape is digitized to a Matroska/FFV1 file at NTSC specifications with YUV 4:2:2 10 bit video and 2 channels of 24 bit LPCM audio
- A 1/4" audio reel is digitized to a 2 channel 96000 Hz, 24 bit audio LPCM file
- A CD-R is ripped to a 44100 Hz, 16 bit, 2 channel LPCM file
- A DAT tape is ripped to either a 32000, 44100, or 48000 Hz 16 bit file

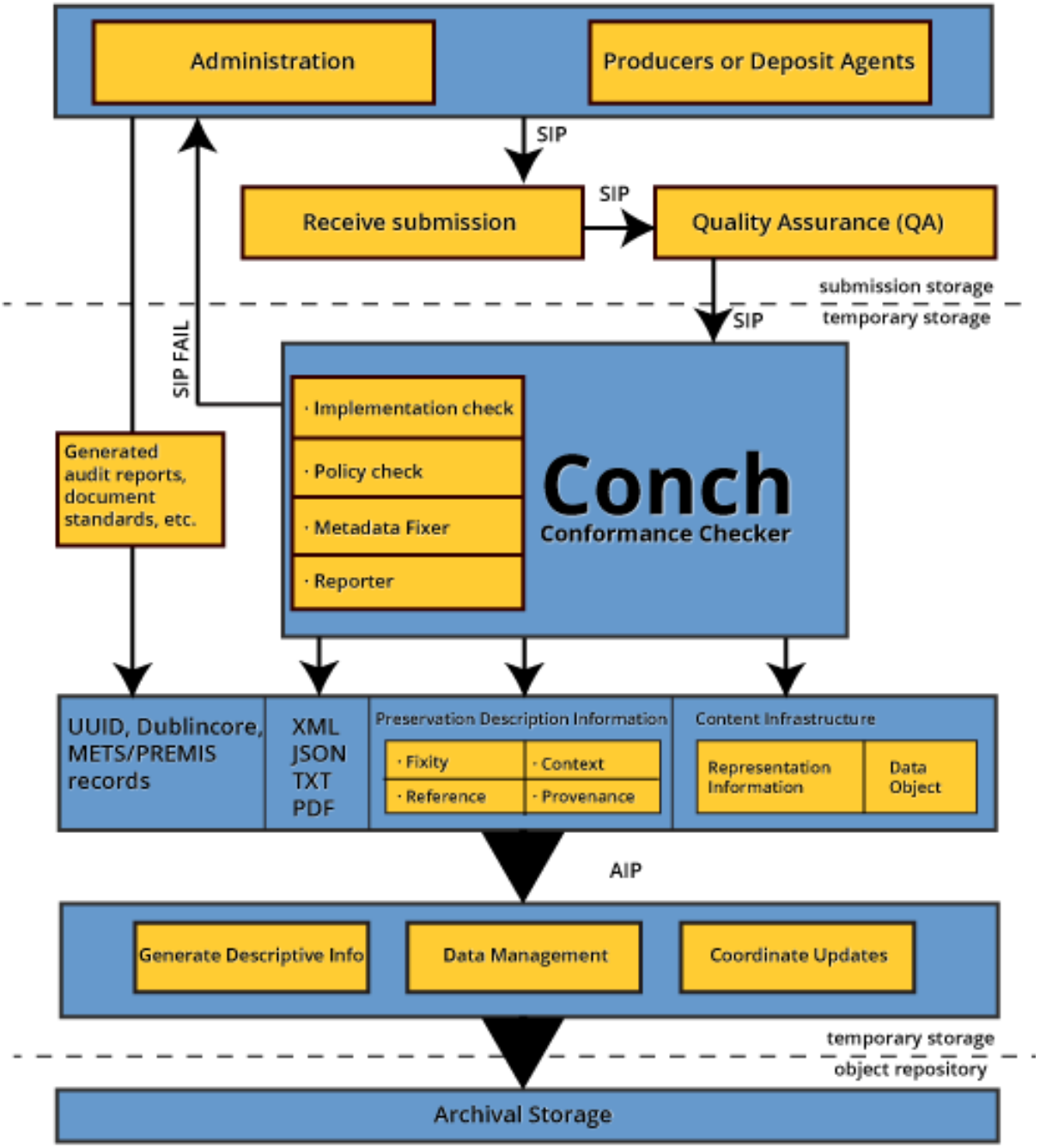


Figure 7: The OAIS Ingest Functional Entity with Conch Integration

Such digitization requirements may be expressed into a policy checker set through the shell or policy checker to verify that the results of digitization are consistent with the archive's specifications. This includes both sets of technical metadata and specification as well as anticipated embedded descriptive, preservation, or administrative metadata.

Verification of Lossless Digitization Until recently audiovisual digitization required a fairly inflexible set of hardware requirements and extremely limited possibilities for an open source approach to video digitization. Due to the bandwidth and processing requirements for the digitization of standard definition video required the installation of PCI cards and often the use of hardware encoders that were designed to encode video as fast as the video was being received to codecs like MPEG2 or JPEG2000. With modern connectivity options such as USB 3 and Thunderbolt it is easier to add video digitization capabilities to modern computers and more archive are performing this internally. Additionally modern computer processers can now transcode video losslessly in software from a video input without the need to rely on proprietary hardware-based encoders. Open source solutions such as DVA Profession, bmdcapture, and FFmpeg along with the open provision of video digitization software development kits, such as the Blackmagic SDK are facilitating new open development projects for archival video digitization.

As vendors and memory institutions are increasing considering and implementing digitization workflows that encode video directly to lossless codecs without the use of an intermediate file-based uncompressed audiovisual data, it is increasingly crucial to assess this lossless file soon after creation to detect any flaws within the digitization process.

For those digitizing video through processes that incorporate libav or FFmpeg such as bmdcapture of FFmpeg's decklink integration, a separate framemd5 may be written alongside the encoded FFV1 data. The resulting FFV1 data may then be verified against the framemd5 to verify that the correct bits were written to disk.

An inspiration for the use of framemd5 reports within a digitization workflow is inspired by the verify option with the flac utility available at http://flac.sourceforge.net/. The '-V' or –verify command is used to decode the encoded stream in parallel to the encoding process to double-check the losslessness of the transcoding. With this method any discrepancy between what data is read and transcoded versus what data is written to disk could be identified in a subsequent verification process. The use of framemd5 data within a digitization workflow enables verification in cases where an option similar to flac's –verify argument isn't available.

Assessment of Vendor/Producer Deliverables For archives that clarify specifications for audiovisual digitization projects, the implementation checker should facilitate a workflow for the archivist to express those specifications and verify received material against them. In addition to testing for the presence and order of required metadata tags the implementation checker should also be able to verify that they adhere to particular patterns as expressed through regular expressions.

The implementation checker should be able to verify that files were transferred completely and that the delivered material does not contain any partial files from an incomplete or aborted transfer.

The implementation and policy checker's reporting on deliverables will enable the user to provide specific feedback to the vendor or producer to create files with greater compliance or coherency.

Conformance Checking at Migration Time

Fixity Verification Migration of large amounts of data introduces risk for digital corruption and/or sector loss. Ongoing data migration is essential for digital preservation but can require a time consuming verification process. Both Matroska and FFV1 contain features for internal fixity so that a file copied from point A to point B can be assessed at point B alone to verify the data integrity of the frames. MediaArea recommends using Matroska's CRC features for use in digital preservation to allow for fixity verification to be more stable and achievable with the file alone without necessarily depending on external databases or records of checksums.

Obsolescence Monitoring Migration is typically an ideal time to perform obsolescence monitoring and preparing actions to limit complications in obsolescence status. Just as memory institutions must maintain the technology that their physical collections are dependent upon, this is equally true for digital collections. As this maintenance becomes more complex, costly, or unlikely archives will typically reformat material (with as little compromise to the content and characteristics of the source as possible) to a format that has more sustainable characteristics.

To counteract arising obsolescence challenges it is critical to have access to thorough sets of technical metadata in order to associate certain codecs, formats, or technologies with sustainability risks or to identify what one format should be superseded by another in a particular digital preservation. For instance an institution that utilized FFV1 version 0 as a lossless preservation codec may wish to identify such files to reformat them to FFV1 version 3 (now that it is non-experimental) in order to take advantage of version 3's additional advantages. In our research one archive found that some digitized material received from a vendor was missing technical metadata about field dominance and had to identify exactly which materials were affected to order to rectify the issue.

The team and roles

- Jérôme Martinez (Digital Media Specialist): technical design, implementation of the bytestream/bitstream analyzer, extraction of metadata.
- Guillaume Roques (Back end / Front end developer): database management, automation, performance optimization, shell.
- Name to be confirmed (Junior developer): GUI development, reporting.
- Dave Rice (Archivist): communication with memory institutions, definition of tests, documentation.
- Ashley Blewer (Archivist): technical writing and documentation, design and user experience optimization
- Tessa Fallon (Archivist): technical writing and documentation, community outreach, standards organization
- Erik Piil (Archivist): technical writing and documentation, OAIS compliance support

Community

Artefactual Systems and Archivematica

Artefactual Systems is a privately owned company incorporated in the Province of British Columbia with expertise in open-source, open-standard technologies for archival collections and digital repositories. Artefactual is best known for its two open-source software tools, the Archivematica digital preservation system and the AtoM online access system.

MediaArea proposes to include within its project a component focused on implementing parts of the MediaArea conformance checker within an independent and OAIS-focused repository system. As many of Archivematica's development philsophies (modularity, OAIS, open source, and focus on memeory institutions) align well with the spirit of PREFORMA's Challenge Brief and Archivematica was an early adopter of Matroska/FFV1/LPCM within a preservation context, we have selected Artefactual as an ideal collaborator to ensure that the results of our work are well-prepared for integration within existing open source repository solutions. Archivematica can benefit from PREFORMA's development of conformance checkers to strengthen that step of the OAIS processs. Additionally the incorporation of the conformance checker into Archivematica shall allow memory institutions with a new means to access the results of the project.

We believe that incorporating Artefactual into our phase 2 proposal provides a meaningful deliverable, the incorporation of a PREFORMA conformance checker into a key OAIS solution. The collaboration also provides the team with a strategic and indenpendent test case implementation. Although Artefactual will not take part directly in the development of the conformance checker, our project proposal includes funding to sponsor Artefactual to test, provide feedback upon, and implementation selections of the conformance checker. Additionally we anticipate that Artefactual's existing work on the Format Policy Registry may provide a positive influence on our work on the policy checker.

Project Advisors

MediaArea has approached several individual and institutional partners from varying types of organizations to provide expertise, testing, and feedback to the project as it develops. The intent of this aspect of our proposal is to facilitate an efficient and responsive mechanism for feedback with specialized areas of expertise relavent to the project. This initiative also kickstarts the relationship between the project, open source development communities, and PREFORMA's target users. Such partners would receive a fixed stipend amount from our proposed in exchange for participation in the projects mailing list, occasional requested meetings, and commenting on the project.

- Moritz Bunkus (Matroska main developer): validation of Matroska tests, Matroska specific technical support, review of standardization efforts
- Michael Niedermayer (FFmpeg maintainer and FFV1 primary author): validation of FFV1 tests, FFV1 specific technical support, review of standardization efforts
- Luca Barbato (Libav maintainer): validation of FFV1 tests, FFV1 specific technical support, review of standardization efforts
- Ian Hendersohn (User of FFV1/mkv/lpcm in a national archive): Provide testing and feedback of project tools
- Peter Bubestinger (User of FFV1/lpcm in an audiovisual archive): Provide testing and feedback of project tools
- Artefactual Systems (consulting and development company): Provide testing and feedback of project tools, implement select conformance checker tools into Archivematica
- Kieran Kuhnya (encoding strategist): Advisor on implementation and standardization efforts

Through real-time feedback and specialized review these participants will provide crucial evaluation of the project throughout the second phase.

Open Source Sponsorship

Through MediaArea's project plan there are several areas where we believe the resulting implementation and use of the project deliverables would benefit from the sponsoring of related improvements through key existing open source projects. To this end, MediaArea is reserving a portion of its overall budget towards sponsoring development in external projects. These funds would be used when it is more efficient to time and project cost to sponsor a small development rather than have our own team create the patch.

One specific receipent of such sponsorship would be as bounties within the issue trackers of ffmpeg, libav, vlc, and/or other related open source projects. For example, our plan anticipates supplying recommendations to next version of the Matroska file format, but such additions to the specification are not meaningful to the user community without their implementation in the key tools for reading, writing, and adjusting matroska files. Thus if working through matroska-devel we successfully present a patch for adding field dominance or color matrix expressions to increase Matroska's ability to self-describe than such funds will be available to the project implement the use of the features.

Example of usage in European Memory Institutions

The National Library of Wales (NLW) has used MediaInfo in their digital audiovisual preservation workflow for several years.

Former Chief Technical Officer of the National Library of Wales said: "As a National Library incorporating the National Screen and Sound Archive of Wales, we have to preserve digital audiovisual material in perpetuity. Part of this work is characterising AV files and extracting technical metadata. We found no better tool at this job than MediaInfo, and the support and response from MediaArea SARL has always been excellent."

Vicky Phillips, Digital Standard Manager, said: "The National Library of Wales has been using MediaInfo as a technical metadata extractor tool since we started preserving Off air (television and radio) recordings in 2009. From 2009 up until last year we were using an internally developed transformation stylesheet (xslt)

in order to map the output from MediaInfo to the Library of Congress audioMD and videoMD metadata standard. This metadata was then stored alongside other administrative and descriptive metadata for that object within a METS document within our Digital Repository. At the time this worked quite well for us, although we had to fix any issues with the xslt when changes were made to MediaInfo output, which caused our transformation to fail or produce errors. However, we did feel that this localised schema mapping implementation wasn't a very sustainable solution. So when we were looking at updating the system which captures our Off-air recordings last year and were looking at reviewing workflows and metadata etc. we decided that this would be a good opportunity to try and get the technical metadata we required for preservation purposes to be output directly from MediaInfo. The Library of Congress audioMD and videoMD were always seen as just a stop gap until other metadata schemas were developed. We therefore started looking at both PBCore and EBUCore. EBUCore's elements seemed a lot more specific than PBCore and being an European standard which is utilised by a number of European projects it was felt that this was the best option for us. Discussion between MediaInfo, EBUCore and myself then commenced in order to produce EBUCore directly from MediaInfo. With the aim being of developing a mapping which is developed and maintained by those who are involved with the software and metadata standard (developers and users). This would then enable the audio visual community to be able to save technical metadata to a globally recognised metadata standard without having to do any mapping themselves. We also use MediaInfo as a validation tool too (similar to JHove). For example we use it to check that the duration of the digital file isn't empty. If it's empty we know that there is an issue with the file and somebody takes a look at it and sometimes the item is re-processed. We have been running MediaInfo on MP2-TS files from 2009 and are planning on running it on DPX, ProRes and IMX files very soon."

MediaArea demonstrated his capability to understand the need of the memory institution, to "translate" it in a technical implementation, and to discuss the project with a standardization institution.

Open Source Ecosystem

Cross Platform Support

MediaArea excels in open source development for cross-platform support and chooses development frameworks and tools that enable cross-platform support to be maintained. Several applications developed by MediaArea such as QCTools, MediaInfo, and DVAnalyzer are available under nearly all major operating systems. To achieve this we will program in C++ and use the Qt application framework (only for the GUI, pending licensing).

For an impression of MediaArea's focus on cross platform usability please see our download pages:

- http://mediaarea.net/en/MediaInfo/Download
- http://bavc.org/qctools-downloads

MediaInfo is also officially provided by multiple open source distributions:

- Debian: https://packages.debian.org/wheezy/MediaInfo
- Ubuntu: http://packages.ubuntu.com/utopic/MediaInfo
- RedHat / Fedora: https://apps.fedoraproject.org/packages/MediaInfo
- OpenSuse: http://packman.links2linux.org/package/MediaInfo
- Arch Linux: https://www.archlinux.org/packages/?q=MediaInfo
- FreeBSD: http://www.freshports.org/multimedia/MediaInfo/
- Homebrew (open source software package management system for Mac): http://brewformulas.org/MediaInfo

Online Resources

MediaArea will utilize GitHub as a social and development center for Conch development and uses GitHub's issue tracker and wiki features alongside development.

For communication MediaArea will establish public mailing lists and an IRC channel for foster support and involvement from memory institutions.

MediaArea will solicit, create, and accept test files and reference files that highlight various features of the implementation checker and illustrate likely preservation issues that may occur within the selected formats.

Community Interviews

In December 2014, MediaArea started conducting interviews with FFV1, Matroska, and LPCM stakeholders in order to collect feedback and insights from the archives community. To date, interviews have been conducted with:

- Hermann Lewetz, Peter Bubestinger; Österreichische Mediathek
- Ian Henderson; UK National Archives
- Christophe Kummer; NOA
- George Blood; George Blood, L.P.

Notes and partial transcripts (in English) from the interviews are available in the MediaInfo PREFORMA GitHub repository. Public release of interviews is pending complete transcriptions and review of transcriptions by all participants in order to ensure accuracy and compliance with Creative Commons CC-BY 4.0. The interviewees' feedback will help inform MediaArea's approach to development in all areas, and especially reinforced our plans to standardize the FFV1 specification through an open source standards organization

Advance Improvement of Standard Specification

FFV1 Specification Efforts to create an FFV1 specification began in April 2012, continuing through the August 2013 release of FFV1 version 3. Currently the specification remains in development at http://github.com/ffmpeg/FFV1. Ideally a specification should fully inform the development of a decoder or parser without the need to reference existing implementations (such as the FFV1 implementations within ffmpeg and libav); however MediaArea's initial research and prototyping efforts with FFV1 found the current specification insufficient to create a decoder. As a result MediaArea utilized ffmpeg's FFV1 implementation to fully interpret the specification. Several threads on the ffmpeg-devel and libav-devel listserv reference discussions about the development of the FFV1 specification and consideration of efforts to standardize the specification through a standards organization, such as IETF (Internet Engineering Task Force) 1.

In consideration of FFV1's utilization within preservation contexts, the standardization of the codec through an open standards organization would better establish FFV1 as a trustworthy, stable, and documented option. In MediaArea's interviews with FFV1 adopters, interviewees noted that FFV1's current status proved problematic in gaining organizational buy-in for adoption of FFV1. Additionally, standardization of FFV1 would increase awareness of and interest in FFV1. This increased visibility is vital to engaging an overly cautious archives community. At the moment FFV1 can be seen at a tipping point in its use within preservation context. Its speed, accessibility, and digital preservation features make it an increasingly attractive option for lossless video encoding that can be found in more and more large scale projects; the standardization of FFV1 through an open standards organization would be of broad interest to digital preservation communities and facilitate greater accessibility of lossless encoding options that are both efficient and standardized.

MediaArea proposes working closely with the lead authors of the FFV1 specification in order to update the current FFV1 specification to increase its self-reliance and clarity. Development of the FFV1 specification early within the PREFORMA project will generate substantial feedback to the authors of the specification which could then be offered through the specification's github page via pull requests or the issue tracker. MediaArea proposes at a later stage of development that the PREFORMA project serve as a catalyst to organize, facilitate, and sponsor the IETF standardization process for FFV1.

Considering the two-year timeline of the PREFORMA project and usual pace of IETF standardization projects, we propose at least submitting FFV1 as an Independent Submission to IETF that could

provide workable timeline, encourage a detailed review process, and assign a formal RFC number to the specification.

Matroska Specification Both the Matroska specification and its underlying specification for EBML are at mature and stable stage with thorough documentation and existing validators, but several efforts of the PREFORMA project can serve as contributions to this specifications. The underlying EBML specification 2 has already been drafted into RFC format but is has not yet been submitted to IETF as an Independent Submission or otherwise. MediaArea recommends that PREFORMA play a similar catalyst role for further standardization with Matroska as well, helping enable the refinement of the current RFC draft and coordinating an IETF process.

Matroska has a detailed metadata specification at http://www.matroska.org/technical/specs/tagging/index.html. Each tag has an official name and description while provides rules and recommendations for use. Many of these tags could be associated with validation rules, such as expressed by regular expression to assure that the content of the tag conforms to expectations. For instance tags such as URL, EMAIL, or ISBN have specific allowable patterns for what may be contained. As part of build a conformance tool for Matroska, MediaArea will generate conformance tests for individual tags and these tests may be contributed back to the Matroska specification in a list of regex values, an XML schematron file, or other acceptable contribution method.

Other Suggested Improvements or Contributions to Standard Specifications

- Register an official mime type via IETF for Matroska.
- Register dedicated FFV1 codecid with Matroska (current use is via fourcc)*.
- Proposal of a tagging extension to Matroska based on the requirements of the digital preservation community.
- Feedback for features and functions of FFV1 version 4, which is currently under development.
- Creation of metadata translators to convert common descriptive metadata formats within memory institution. For instance convert EBUCore into the XML representation of the Matroska tagging specification so that such metadata may be easily imported and exported between EBUCore and Matroska.
- fourcc is AVI-style with some bitstreams not having a clear license due to coming from Microsoft

Sustainable Open Source Business Ecosystem

MediaArea has long been an open source native and has an open source business model based on sponsored support (bug correction and feature requests), application support, and branched customization based on an institution's specific needs since 2007. Previously existing in a non-business capacity since 2002.

MediaArea's long term goal is to merge previous open source standalone products designed specifically for broadcasting and memory institutions into its flagship product, MediaInfo. These products include the WAV implementation checker, professional metadata editor and fixer BWF MetaEdit; the AVI implementation checker, professional metadata editor and fixer AVI MetaEdit; and the baseband analyzer for quality assurance, QCTools. Each piece of aforementioned software, designed by MediaArea, has a strong focus on individual areas of digital preservation based on the specific sponsor's needs. Thanks to our discussions with memory institutions, we strongly believe that an integrated environment for conformance checking is sorely needed in the field. By sponsoring the Matroska/FFV1/LPCM + shell/Implementation Checker/Policy Checker/Reporter/Metadata fixer parts of this project, PREFORMA plays a major role in the creation of a fully integrated and open source implementation checker.

MediaArea plans to build this stable, integrated solution over the course of the PREFORMA project phase, which will include the current team investigations of Matroska, FFV1, and LPCM, as well as other PREFORMA investigations such as TIFF and JPEG-2000. This will ensure that proper feedback from PREFORMA developers and stakeholders is provided in a meaningful timeframe. After the PREFORMA

project is completed, MediaArea anticipates offering access to an integrated solution in two ways: as a ready-to-use environment with a subscription business model (SaaS), and as a ready-to-download version of the integrated solution. This is based on MediaArea's future business model, which consists of a combination of subscriptions and paid punctual support, such as bug corrections and new feature requests. With this long term business model approach in mind, MediaArea will be able to continue offering a PREFORMA-specific version, free of non-PREFORMA related layers, as a subset of our own integrated solution.

Participation at Open Source conferences

The MediaArea team is active in the open source community and has presented the work on PREFORMA at two conferences during Phase 1:

- Dave Rice presented our work on PREFORMA at FOSDEM on January 31st:
- https://fosdem.org/2015/schedule/event/enabling_video_preservation/
- Ashley Blewer presented our work at Code4Lib on February 11th:
- http://wiki.code4lib.org/2015_Lightning_Talks

Project Management Strategy

Goal:

To ensure a vital project, the MediaArea.net team will track processes through an open issue tracker, allowing for consistent and detailed reports with an emphasis on feedback and transparent communication throughout various iterations of the project.

Method:

It is through daily task distribution, management and reflection that project advancements, as well as risks, are addressed. Such open, community-based interaction in the management and implementation of the project allows evolution to occur in all components of the endeavor to include: management, administration and documentation with the allowance for other opportunities for discussion and change to emerge throughout the whole of the project. In the process, the MediaArea team will provide detailed project reports that encourage open, constructive feedback during the testing process that will shape and influence discussions during project meetings.

Justification/Purpose:

Such assessments, occurring daily, aim to build a sense of community that can freely and continually address all deviations and uncertainties, highlighting changes and assessing the benefits, as well as problems with each shift in the projects testing and implementation. With daily communication, evaluations, critiques and suggestions can be addressed with quickly and efficiently.

Intended Result:

This approach to risk management is well-established within The MediaArea team's previous work, with openness between all involved being strongly encouraged. The MediaArea hopes for and invites all relevant communities involvement in the strategy and management of the project, developing and including use of public tools (github.com, wikis, forums, etc.) and other communication during the planning and implementation of the project timeline. Such interactions and testing of software are welcome at any point during the implementation stage. These engagements will help immensely in addressing and evaluation of priorities and outcomes of the project. Through a process of constant assessment and open access the MediaArea team can respond to changes in software and address performance and usability of the software in its various forms.

Risk Analysis Model

- Define Risk
- Establish date of risk
- Define risk (factors, causes and possible effects)
- Include other contextual information related to risk
- Record individual who discovered risk
- Establish risks probability and impact
- Hypothesize time when risk could occur
- Hypothesize the impact of risk
- Prioritize risk based on established effects of risk
- Classify risk
- Establish point-person for addressing risk
- Create method to reduce likelihood of risk or avoid risk entirely
- Establish alternative plan should risk be unavoidable
- Mark last emergence of possible risk
- Declare risk concluded (Responsibility of Project Manager)
- Note date of the ending of risk
- Contextualize in writing
- Share with pertinent parties ** status and date when the status was last recorded, ** person who accepted the risk, e.g. project manager, ** conclusion date and ** reason for conclusion.

Internal Risk Assessment

Internal Management: To assure the flow of the project, the team will establish a team leader, as well as a junior leader, for the major components of the project. In the case of any event in which the team leader is unable to retain their duties, the junior leader will take their place. Though established as a junior leader, the leaders will function as joint team leaders through the entirety of the project, barring the stepping down of either team leader for unforeseen reasons.

In the event of a major change to staff (whether through avoidance of risk to project vitality, or external life events), the remaining team members will communicate about the best method for reallocation of resources to either replace lost staff, or consider changes in distribution of work among the remaining members. All changes will occur openly, with the participation upon relevant groups and stakeholders. In the event of a drastic shift in leadership, the newly appointed person will affirm the roles of leadership newly assigned to them and work to adhere to the standards of the original contract. If changes to the contract are required, alterations will also occur openly, with the participation of relevant groups and stakeholders.

Addressing Unrealistic Schedule In the event that the project appears as though it will move beyond the original scope of time, the team will communicate and establish the reasons for the delay. If these reasons can be addressed and changed to fall within the original schedule necessary changes will be made. Such changes and discussions will remain open to relevant stakeholders. If the team communicates and changes cannot be made to address the deadline, then a decision will occur within the team as to the plausibility of establishing a new schedule. This discussion will remain open and include all pertinent parties. If it is found that a new schedule can be implemented such changes will be made, at which point all contracts, funding schedules and likewise paperwork will be altered.

In turn, should the team complete work before the established date a meeting will occur to discuss what additional work might remain, or how the deliverable products might be further improved. If such improvements are established, the team will continue work within the schedule time. Such decisions will include input from all communities involved. Alternatively, should the team find that their work is at the point of ideal completion, they will meet to discuss how best to end before the scheduled date. This discussion will include all relevant stakeholders, with changes to contracts, financing and other factors included as necessary.

Addressing Unrealistic Budget: Based from the experience of MediaArea's recent software developments projects of similar scale to PREFORMA we are confident that our budget, objectives, and allocation of resources are appropriate and realistic.

In the event that the project appears as though it will exceed the proposed budget, the team will communicate and establish the reason for the change in financial expectations. If these issues can be addressed and changed to stay within the original budgeted amount, such changes will be implemented. The decisions involving these changes will be shared with pertinent parties. If the team meets and decides that the work will exceed the budget, an establishment of the dollar amount by which the project will exceed its original amount will occur. This amount will be shared with all vital stakeholders.

After the team establishes the amount with which the project will run over budget, the possibility for addressing this funding change will occur, such considerations will include all relevant communities. If the new budget is attainable, work will be redistributed with all contracts, payments and financial elements in need of alteration occurring.

Should the team be unable to attain the necessary funding to continue the project. The deliverables on the project at the point of the cessation will be handed over to pertinent projects. Considerations about continuing the project will be returned to if financing opportunities emerge.

In turn, should the team complete the project under the established budget, a discussion will occur as to how the remaining monetary commitments might be used to improve or expand upon the projects goals. Such discussions will be transparent and include all parties involved. If new budgeting occurs, changes to all necessary contractings and financial paperwork will occur accordingly.

Ceasing of File Format: Should any file format become incompatible the team has the ability to analyze and address any existing format and the changes that occur with the respective format. The project software is designed based on past project experiences, to allow for movement between different formats. The team has already dedicated research and development efforts towards standardizing formats to specifically assist in the archival sustainability of the MKV and FFV1 formats.

Additionally, the software framework can easily be applied to analyze any file format with minimal change. The software is developed so that it is not difficult to swap in other formats and policies, and the MediaArea team has experience doing such with MXF or MOV file formats, should the Matroska format no longer continue development.

Software Incompatibility: Should the software become incompatible due to major operating system updates, API conflicts, or other unforeseeable technical issues, the team has the ability to analyze and address any existing software and the changes that will occur with the respective software. The project software is designed based on past project experiences, to allow for movement between different software structures. For example, if a major operating system such as Windows changes intrinsically and does not allow for the software to run at its basic level, crucial updates will be patched as needed. The modularity of the project software allows for key components to work independent of each other, making it even more unlikely that a system-wide failure should occur based on software compatibility. The software is expressly written in C++ due to its longstanding compatibility standards and optimal portability between systems. Notable changes to software compatibility will be shared with pertinent stakeholders.

Open Source Compliance: Should the supporting software chosen as foundational software for the creation of this project become unavailable due to open source licensing complications, an alternative open source solution will either be chosen to work in its place, or an in-house solution will be developed, depending on if an alternative solution exists or if budget constraints allow for the creation of in-house software components. Timing and budget priorities can be assessed, reviewed, and implemented according to need. All changes will be discussed and considered with the input of relevant communities. # Introduction to Architecture

This serves as a roadmap for the technical components of the project, split into two categories: Global architecture and Checker architecture. The global architecture schema defines the context in which the PREFORMAMediaInfo software is situated and gives a high-level understanding of the software. The Checker Architecture details the structural components of the conformance checker and metadata-grabbing module.

The conformance checker's goals are based on the following core principles:

Applicability

The conformance checker will provide essential services to the functional entities described in the Consultative Committee for Space Data Systems (CCSDS) and International Organization for Standardization (ISO) reference model for an Open Archival Information System (OAIS), intended for the long term preservation of digital information.

Portability

The checker has the capability to be packaged and run as an executable on a computer running any common operating system. For this reason, the shell has plans to be integrated into three platforms: Command line, graphical user interface, and a web-based (server-client) platform. Qt was chosen as the core toolkit for the development of the graphical user interface because of its flexibility and ability to be deployed across many different operating system platforms.

The developed software will also have the capability to function as a micro-service application alongside other digital preservation systems such as Archivematica. With its micro-service approach, Archivematica serves as a wrapper for related task-specific software and is an open source system that also works to maintain standards based on providing access to digital collections. The conformance checker suits this systems' design as an integrated micro-service suite, allowing for it to run alongside other third-party software tools that also serve to process digital objects and help to standardize a preservation focused ingest-to-access workflow. Within this design, various micro-services, including the conformance checker, can be built together into customized workflows and deployed across all operating systems and platforms.

As the shell will be developed to integrate into and perform within a web-based platform, the web user interface (UI) will function across 3 major web-based browsers. These platforms are to include Firefox, Google Chrome, and Internet Explorer. The software's UI and capabilities will function similarly across all of these relevant web-based environments.

Scalability

Similar to the way in which MediaInfo can be built and expanded upon in archival institutions to perform media analysis at scale, the comformance checker can be integrated into scripts and systems that can validate files en masse and deliver computer-readable and human-readable metadata via standard XML reports.

The scalability of each built component of the conformance checker will allow the software to scale horizontally throughout multiple computers and servers in order to potentially meet a heavy and increased workload. The software can be deployed on more computers for added speed, dexterity and reliability, distributing and partitioning tasks to a cluster of machines in order to improve performance and increase the working capacity of the software. Built within this architecture, the conformance checker can be expanded upon and integrated into various workflow environments with the added ability to increase its number of file requests.

Distribution

The source code will reside on an open development platform (Github) in order to provide easy access and distribution during all stages of development. It will also have the ability to extract nightly builds of the code and deploy using continuous integration. The subsequent builds and software releases can then be downloaded, built and run on any system.

Modularity

MediaArea plans to collaborate with the other PREFORMA teams to support optimal interoperability with each other as well as with third-party developers of additional conformance checkers that will utilize

the implementation checker shells. The checker's API allows for the checker to be integrated alongside other components and the future development of plug-in features. The architecture's modularity will also allow the software's features ongoing and improved maintenance as well as offering reliability as a freestanding shell.

The open source project will be programmed and provided in C++ but will be functionally compatible with other programming languages through the use of bindings. Other than C++, the team will support bindings to C, Python, Java, C# and other languages upon open request and feedback. The bindings will function as wrappers to allow the original code to be explicitly used and supported within other common languages. Supporting this capability will allow the API to be fully integrated within the open source community with the full potential of interoperability.

Deployment

The shell will be deployed on the PREFORMA website, as a stand alone shell, networked with other PREFORMA shells currently in development and used within test environments. Due to the levels of interoperability set up as an integral component of the conformance checker, it can be utilized within legacy systems as well as future systems. The shell will also be released to run on all relevant deployment platforms, implying that any user will be able to download and operate the most up-to-date version of the conformance checker.

Interoperability

MediaArea's API will allow for the full integration and interoperability between all software systems. The API will be developed to operate between the targeted systems while offering the ability to run and behave similarly across each, whether the user is operating within MS Windows, Mac OSX, or Linux distributions.

To enable growth within an open source environment, it is essential for the API and all corresponding data structures and software releases to be able to comply to and function within an interoperable architecture across different deployment platforms and software systems. System-to-system interoperability can be enhanced to meet user's potential needs, and the conformance checker's API will help implement an effective exchange and integration of the required data and information. This integration will suit other software and will work across whichever system the user is running the software. The conformance checker's shell will also operate within legacy and future systems.

Global Architecture

This includes the following:

- Technical specifications
- Open source conformance compliancy
- Relationships between frameworks
- Framework traversal patterns

This is broken down by category and the categories are as follows:

- Common Elements
- Core
- Database
- Scheduler
- New file daemon
- implementation checker and metadata grabbing module
- Policy Checker
- Reporter
- Shell(s) ** CLI (Command line interface) ** GUI (Graphical user interface) ** Web (Web based interface)

Checker Architecture

This includes the following:

- Technical specifications
- Relationships between each structural component
- Relationships between plugins

This is broken down by category and the categories are as follows:

- Transport interface
- Container/wrapper implementation
- Container/wrapper demuxer
- Stream/essence implementation
- Stream/essence decoder (through plugin)
- Stream/container coherency check
- Baseband analyzer (optional, through plugin)# PREFORMA Global Architecture

PREFORMA strives to offer access and ease to users with a structure that operates similarly and efficiently cross-platform and functions in both an online and offline capacity within different interfaces.

Architecture schema

Common to all elements

All elements can be installed on the same server or on different servers. This will be dependent on the expected workload and anticipated points of access. The flexibility of the architecture allows for a single user to either use the shell on one computer or to use it on multiple computers, depending on scalability and need. A distributed system can be set up to allow the implementation checker to process large files more quickly than when working on a single machine. This architecture offers this ability with relative ease for the user. In addition to added speed, this build of scalability within the implementation checker increases its capacity to function reliably within various anticipated workflows.

File access

File access using CLI To place a file into the checking system, a user may execute the CLI name file manually or they can asynchronously run the file and test for its readiness. If the files are passed in a batch format, the reports can be later retrieved by manually requesting updates from the Core.

File access using GUI To place and use a file through the checker's GUI, both technical and non-technical users will work within windows and dialog boxes that enable them to navigate to, open, and load system files directly within the interface. A standard open dialog box will search for and load files for processing within the software. The GUI runs within a human-readable, interactive environment with visually intuitive access to the file checking process. In addition to allowing direct access to checking individual files, scheduled batch file checking via special folders residing on the user's computer/network can also be enabled directly within the GUI.

File access using Web To place and check files through the web-based browser platform, the process will be similar to working within the implementation checker's GUI. In this case, file loading will function visually, as with the GUI platform, but within the specific web browser's user interface (UI) and without the user having to directly download the software. Local network access will power the checker within the browser (Firefox, Chrome, or IE). Similar to the CLI and GUI, the web-based (server-client) platform allows for individual file loading and checking (via a visually intuitive designed interface) as well as enabling the capability of calling up batch-scheduled checks directly via the UI.

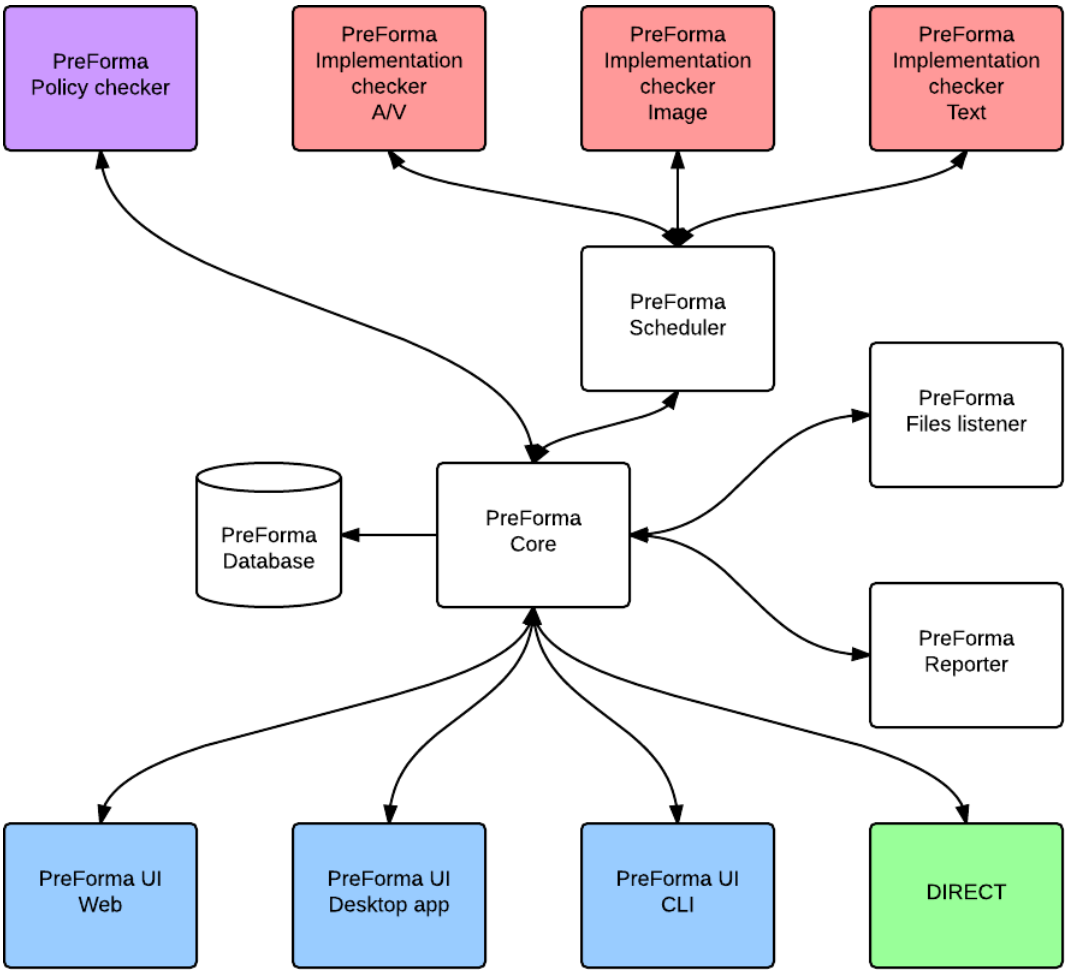


Figure 8: Global Architecture Schema

Common to all Files can be copied to a special folder somewhere on the user's computer or local network. This folder can then be set to regularly check for new files at defined intervals and process the files accordingly. This function can be set up within any of the three checker access methods (CLI, GUI, and Web UI).

File processing

The architecture allows for two modes of file processing: Direct access and asynchronous. Through enabling both processes, users can choose to have the checker handle individual files or have it process them in batches. Direct access allows for files to be processed one at a time, while asynchronous allows for files to be processed in batches and returned to at a later time.

Internet Access

The architecture does not require direct internet access. Users can work with any version (CLI, GUI, and Web UI) offline as long as it is installed on a user's machine (or distributed machines). In this case, for each deployment platform the version is installed and being used on, the user will be able to create an executable even when they are not connected to the internet. The web-browser UI can be run using a local network access.

Automation

PREFORMA includes the option of automated checking, allowing for users to receive notifications when new files are placed or come into the system.

A user can set up a system on the Core which will configure the system to run scheduled automatic checks for new files and batches. This system can be accessed and configured within the user interface to run on a timed and defined schedule.

Batching

A user also has the option to set up batch file checking and validating and schedule it to be sent through to the checker at any later point in time. These large tasks can be set up to perform according to a defined schedule. All versions of checker access (CLI, GUI, and Web UI) will have the capacity to enable this function, either through the command line or directly through the interface.

Prioritization

A user can also prioritize the checker to queue individual items and scheduled checks based on a defined priority level, with a lower priority placed on periodical checks. This function will be available through all versions of the checker (CLI, GUI, and Web UI). Priority levels for checks can be divided by High (for checks requested by user), Normal (for automated checks), Low (for periodical checks) requests.

PREFORMA allows users to load and edit configurations within a REST API configuration that will run via HTTP.

The Web-based system allows users to work both on and outside of the network as well as locally if they choose. If the user chooses to work locally, they will have access to the application server directly. This operation runs via a HTTP Daemon.

Users will have access to this web-based user interface for basic management. This will allow users to see the list of files, as well as the processes and checking happening to each of these files. Users will also be shown demarcation of which files pass testing successfully. The web user interface (UI) will function across 3 major web-based browsers (Firefox, Google Chrome, and Internet Explorer).

Core (Controller)

The Core serves as communication between all plugins within and outside of the Conch system and between all layers. The Core is the main service and runs in a passive, background mode. For example, if a user updates The Core, it will have no effect on the functionality of other systems. If a user begins using MySQL while running the implementation checker and decided to change to PostgreSQL, the Core could be adapted to address such a change. In essence, while components shift, the Core functions to present the data to all databases consistently and similarly and can adapt to different components.

The Core has several major functions:

- controls the checkers and manages data for the User Interface
- waits for commands from the Files listener and User Interface
- sends commands to the scheduler for file-checking
- launch periodical checks
- communicates with the database to store and retrieve data from the checkers
- sends data to DIRECT

The Core supports the following requirements:

- Scheduling
- Statistics
- Reporting
- User management
- Policies management

Interface :

- Scheduler : Advanced Message Queuing Protocol
- Policy checker / Files listener / User Interface / DIRECT : REST API
- Database : native driver

Programming language : C++

Database

The Database is responsible for storing the associated metadata and results of the implementation checker, including the policy checker rules and the user rights management. All specific technical metadata and conformance checking results for each type of file format are sent back to the Core before being stored within the database.

- store metadata and results of the implementation checker
- store the policy checker rules
- user rights management
- trace (optionally)

Interface :

- Core : native driver

Software :

As the main purpose of the software build is to store flat datas, it's more suitable to use a document oriented database (NoSQL). However, a more traditional relational database can also be used.

There are various potential database management system options, contingent upon the open source licensing requirements:

- Relational database : MySQL (GPLv2) / PostgreSQL (PostgreSQL License) / SQLite (Public domain)
- Non relational database : MongoDB (AGPLv3) / Elasticsearch (Apache license 2)

Scheduler

The Scheduler element is a form of software "middleware" that distributes the files to be checked across the implementation checkers by using a message broker interface. It translates the file data into one unified language for access within all aspects of the software. The Scheduler also controls priority levels for file checking.

- distributes files
- translates file data into unified language
- batch processing
- priority

The scheduler can take care of the priority function within the implementation checkers : * high : for checks requested by user * normal : for automated checks * low : for periodical checks

Interface :

- Core : Advanced Message Queuing Protocol
- Checkers : Advanced Message Queuing Protocol

Software :

RabbitMQ (MPL 1.1) / Gearman (BSD) / ZeroMQ (LGPL v3)

Files listener

The Files listener is a background process that listens for new files available for checking and validating. Automated checking, set up through the software, will notify when new files come into the system. Each time a new file is available, or if a file is modified, an event is sent to the Core which automatically requests a check.

Different solutions can be implemented depending on the file's storage and operating system. Implemented solutions can include an inotify notification system API for a Linux kernel or kqueue/kevent for a BSD kernel or files directory scanning.

- Automated checks

Programming language : C++

implementation checker and Metadata Grabbing Module

This module can utilize one or more checkers for each media type. As the implementation checker's process could potentially run for a long time, we use an asynchronous system based on a messaging system in order to not lock up the whole process. Metadata and conformance checking results for each file are sent back to the Core to be stored within the database.

- runs the conformance tests for the different types of media files
- grabs metadata (used for policy checking)

See Checker Architecture for more details.

Interface :

- Scheduler : Advanced Message Queuing Protocol

Programming language : C++ for MediaArea, depends on other participants for JPEG 2000, TIFF, PDF.

Policy checker

The policy checker serves to run tests on all metadata grabbed by the implementation checker and metadata grabbing module (the Checker). A vocabulary of technical metadata for each file format and media type will be created for the policy checker's functions.

- runs the policy tests for the different type of media files.

Interface :

- Core : REST API

Programming language : C++

Reporter

Within each of the developed user interfaces there will be ways to export raw metadata and human-readable JSON/XML/HTML/PDF reports after the conformance checking process. The reporter will define and express how a file's checked metadata corresponds to the validation result standards.

- exports a machine readable report, including preservation metadata for each file checked
- exports a report that allows external software agents to further process the file
- exports a human readable report
- exports a "fool-proof" report which also indicates what should be done to fix the non-conformances

The machine readable report will be produced using a standard XML format, implemented by all implementation checkers working within the PREFORMA ecosystem. This allows the reported module to combine output from multiple checker components into one report while also including sub-elements within the report that will address each conformity check. The report will be based on a standard output format that will be made by the consortium.

The human readable report summarizes the preservation status of a batch of files as a whole, reporting to a non-expert audience whether a file is compliant with the standard specifications of the format or institution while also addressing improvements in the creation/digitisation workflow process.

Interface :

- Core : REST API

Programming language : C++

Will use PoDoFo (LGPLv3+) for PDF export

User interface

The User interface (UI) is the shell component that allows direct interaction between users (or other systems) and the PREFORMA components:

- displays test results
- controls the Core
- allows metadata (descriptive and structural) to be edited
- edit configuration (periodical checks, policy checker, user rights)

Conch will provide three different options for a human interface in order to introduce maximum user interaction and flexibility within the implementation checker. These three interfaces are:

- CLI (Command line interface)

A command line interface will be functional on nearly all types of operating systems and platforms, including those with very little graphical interface support. CLI use allows for integration into a batch-mode processing workflow for analyzing files at scale. This interface is more intended for technical and expert users and for non-human interaction.

- GUI (Graphical user interface)

A Graphical user interface (GUI) will be developed and provided for both expert and non-expert users. The GUI, being based on Qt, has the strength of being versatile between operating systems and does not require additional development time to provide support for multiple platforms. The GUI can function similarly across all deployment platforms.

- Web UI (server/client)

An optional Web-based user interface (UI) will also be provided for both expert and non-expert users. In order to run this option, an internet access will not be needed or required. Local network access will power the checker within the user's chosen web-browser. The web interface will provide access to conformance checks without having to directly download and install the software.

Interface :

- Core : REST API

Programming language :

- CLI : C++
- GUI : C++ / Qt (LGPLv3+)
- Web : PHP/Symfony (MIT)# Checker Architectural Layers

The design of the implementation checker portion of the Conch application will be comprised of several layers which will communicate via a Core controller. The layers shall include:

- Transport interface
- Container/wrapper implementation
- Container/wrapper demuxer
- Stream/essence implementation
- Stream/essence decoder (optional, through plugin)
- Stream/container coherency check
- Baseband analyzer (optional, through plugin)

Transport layer

Conch: File on disk or direct memory mapping

Conch uses the native file API for each operating system to enable direct file access, including files that are still in the process or being written. The inclusion of MediaInfo also offers features for direct memory mapping which will be useful for third-party development or plugins.

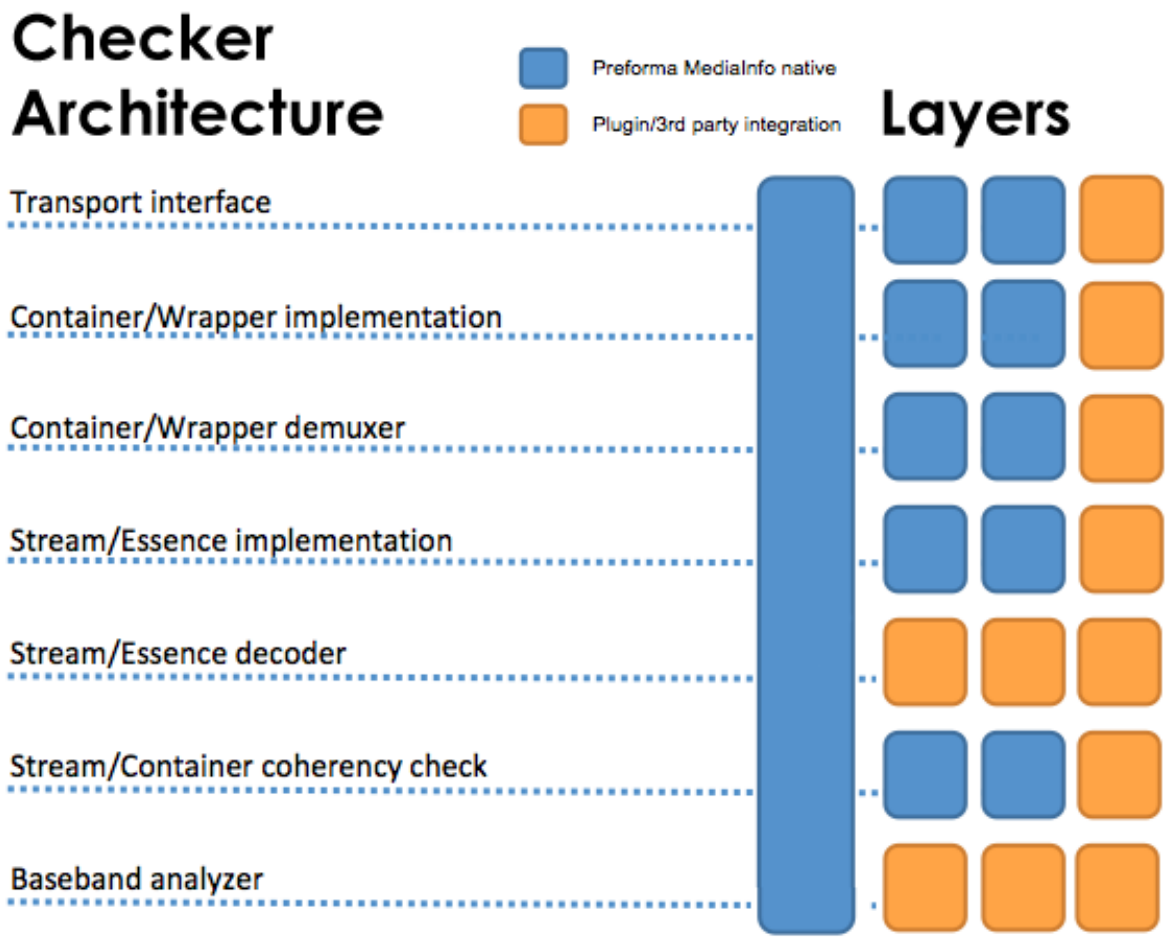


Figure 9: software architecture layers

Plugin integration proof of concept: libcURL

libcURL is licensed under an MIT license that is compatible with both GPLv3+ and MPLv2+. We can relicense to be under GPLv3+ and MPLv2+. curl offers extensive support for transferring data through many protocols. By incorporating curl into Conch the tool will be able to assess files that may be accessible online by providing a URL (or list of URLs) in place of a filepath.

Since we will be generating a library of reference and sample files that will include large audiovisual files, users will be able to assess reference files without necessarily needing to download them.

Used as a proof of concept of plugin integration: HTTP/HTTPS/FTP/FTPS support via MediaInfo open source GPLv3+/MPL2+ and libcurl (MIT license, compatible with GPLv3+/MPL2+)

Container/Wrapper implementation checker

Conch: Matroska checker

Plugin integration proof of concept: mkvalidator

mkvalidator is a basic and no more maintained Matroska checker (BSD license, compatible with GPLv3+/MPL2+) which will be used mostly for demonstration of the plugin integration.

Container/Wrapper Demultiplexing

Conch

Conch will utilize MediaInfo's existing demuxing libraries (can be relicensed under GPL. . . .) which will allow for PREFORMA's selected video codecs, FFV1 and JPEG2000, to be assessed from within many formats founds within archives although these container formats themselves aren't the focus of the current PREFORMA project. Through discovery interviews with archives and vendors we have found FFV1's archival implementations to use a variety of container formats such as AVI and QuickTime as well as Matroska. In order to allow developed tools to support FFV1 even if not contained within Matroska, Conch will support the following formats for demuxing (though not necessarily for conformity (yet)):

- MXF (commonly found within memory institutions)
- MOV/MP4 (often found containing FFV1, JPEG2000, and LPCM)
- DV (video stream format which uses LPCM)
- AVI (used with FFV1 by DV Profession, NOA, Austria Mediathek)
- WAV (a common container for LPCM)
- WAVE64 (64-bit extensions of WAV for 2GB+ files)
- RF64 (64-bit extensions of WAV for 2GB+ files)

By supporting the demultiplexing of these formats through MediaInfo, the developed tools will be applicable to a wide variety of files that contain PREFORMA's selected codecs: FFV1, JPEG2000, and LPCM. This demultiplexing support can be available through MediaInfo's existing libraries in a manner that is compatible with PREFORMA's licensing requirements.

Plugin integration proof of concept: FFmpeg

FFmpeg is one of the most ubitiquous, comprehensive, and open tools for demultiplexing and decoding audiovisual data; however, although FFmpeg's GPLv2+ license is compatible with PREFORMA's selected GPLv3+ license, it is not compatible with PREFORMA's other selected license, MPLv2+. As the PREFORMA conformance project evolves to support additional formats and codecs through plugins the use of FFmpeg's features are expected to becoming more and more appealing.

Although Conch won't incorporate FFmpeg in order to comply with the MPLv2+ licensing requirement, we would like to design plugin support for FFmpeg. In this way a memory institution using Conch could separately download FFmpeg and link the two together to enable additional tools such as:

Stream/Essence implementation checker

Plugin

Conch:

- FFV1
- PCM (including D-10 Audio, AES3)

Plugin integration proof of concept: jpylyzer

For JPEG 2000 (GPLv3+ license, compatible with GPLv3+ but not with MPL2+)

Plugin integration proof of concept: DV Analyzer

For DV (BSD license, compatible with GPLv3+ and MPL2+)

Optional

(Not part of the original PREFORMA tender but can potentially be added upon request after in context of professional services)

- MPEG-1/2 Video (including IMX, AS-07, D-10 Video, FIMS. . .)
- H.264/AVC (including AS-07)
- Dirac
- AC-3 (including AS-07)
- MPEG 1/2 Audio
- AAC
- Any other essence format on sponsor request (we have skills in DV, VC-1, VC-3, MPEG-4 Visual, H.263, H.265/HEVC, FLAC, Musepack, Wavepack, , BMP, DPX, EXR, JPEG, PNG, SubRip, WebVTT, N19/STL, TTML. . .)

Stream/Essence decoder

(Not part of the original PREFORMA tender but can potentially be added upon request after in context of professional services)

Conch

- PCM (including D-10 Audio, AES3)

Plugin integration proof of concept: FFmpeg

FFmpeg decoder (GPLv2+ license, compatible with GPLv3+ but not with MPL2+)

For instance the integration of FFmpeg can provide integration of very comprehensive decoding and demultiplexing support beyond what can be easily provided with MediaInfo's demuxing libraries. FFmpeg's libavfilter library also provides access to waveform monitoring, vectorscope, audio meters, and other essential audiovisual inspection tools.

- Video Waveform Monitor
- Vectorscope

- Ability to inspect luminance and chroma planes separately
- Audio Meters

We anticipate that the implementation of FFmpeg plugin support will substantially simplify the development of other plugins for broader codec and format support so that an entire decoder or demuxer does not need to be written from scratch in order to extend support.

Plugin integration proof of concept: OpenJPEG

OpenJPEG decoder (BSD license, compatible with GPLv3+/MPL2+)

Container/Wrapper vs Stream/Essence Coherency Check

Conch

Conch will support the coherency check between all suppoted formats (see Container/Wrapper implementation checker and Stream/Essence implementation checker parts)

Baseband Analyzer

Conch

- None (only creation of the API)

Playback and Playback Analysis (through plugin)

Note that the PREFORMA tender does not require decoding or subsequent baseband analysis or playback; however, from our experience in implementation checker design with DV Analyzer and QCTools and through discovery interviews, we've found that users are quick to require some form of playback in order to facilitate decision-making, response, and strategies for fixing. For instance if the implementation checker warns that the Matroska container and FFV1 codec note contradictory aspect ratios or a single FFV1 frame registers a CRC mismatch it is intuitive that the user would need to decode the video to determine which aspect ratio is correct or to assess the impact of the CRC mismatch. These layers can be supporting by designing a implementation checker and shell that is prepared to utilize FFmpeg as an optional plugin to enable additional analysis features and playback. Our overall proposal is not dependent on supporting an FFmpeg plugin but we believe that preparing a implementation checker that could support FFmpeg as an optional plugin could create a more intuitive, comprehensive, and informed user experience.

We propose incorporating several compatible utilities into Conch to extend functionality and add immediate convenience for users. Each component is built as a plugin and can be replaced by a third party tool.

Plugin integration proof of concept: QCTools

QCTools graphs (report on and graph data documenting video signal loss, flag errors in digitization, identify which errors and artifacts are in original format and which resulted from the digital transfer based on all the data collected in the past.) ## Implementation Checker

Introduction

The implemententation checker is the centerpiece of the project and where development efforts shall be the most consequential to memory institutions and the underlying standards organizations. MediaArea plans to use the utmost precision and transparency within the development of the implementation checker

so that it may, to the greatest extent feasible, produce an accurate and referenced analysis regarding the adherence or transgression of a selected file format to its associated specifications.

Measuring the extent of the adherence of files to associated specifications can be adjusted by the user, for instance the user may wish to only verify files against a particular version of a specification, or choose to exempt or excuse certain violations, or choose to only assess part of a file (for instance to only test the frame X frames of FFV1 for adherance) in the interest of time.

The design of the implementation checker will enable the user to have some control over the comprehensiveness or extent of the checking process. Additionally the expression of violations should provide both new and advanced users with enough context to the understand the outcome. For instance as the implementation checker reports conformance errors (such as a FFV1 frame expressing an invalid aspect ratio, an MKV file omitting required metadata for an attachment, or that the data size of an LPCM stream contradicts the metadata of the encapsulating format) the output should associate that error with contextual information to reference to violated section of the specification, document potential response, and advise the user to the consequence and meaning of the error.

Ultimately the implementation checker (supported by the other components of the conformance checker) should enable memory institutions to fully understand, validate, and control the file formats selected by the project.

Registry of Checks

The implementation checker must be associated with an active and open registry of checks. The registry shall describe each conformance test in easy to understand, approachable language, with citations to the associated specification, and details of any logic, deductive reasoning, or inferrence clearly detailed. The registry should associate each test to the source code that enacts the so that the human-defined and computer-defined implementations of each check may be strongly linked.

The Registry must allow support community involvement, such as associated forums, issue trackers, links to asscoiated sample files, etc. The design of the Registry and its intertwinement with the Implementation Checker will provide users with a means to navigate from their discovery to sample files, source code, community knowledge, and context.

The registered checks shall be categorized according to its associated authority and specification document. The grouping of rules shall enable the user to select parts of an implementation check if necessary. For instance the Matroska specificiation is comprised of several versions and several underlying specifications, such as Extensible Binary Meta Language (EBML). The implementation checker should be able to test if a file is valid according to all known checks asscoiated with Matroska, but should also be able to validate a file against only the rules derived from the Extensible Binary Meta Language specification.

Demultiplexing

MediaArea's project focus includes two formats of encoded audiovisual data (FFV1 and LPCM) that are usually found within other container formats (which may or may not be Matroska). We intend the implementation checker to function properly for FFV1 and LPCM even if they are not contained within Matroska. MediaInfo already contains the ability to parse encoded streams from container formats such as AVI, QuickTime, MXF, and many others. By re-licensing MediaInfo under PREFORMA's licesning requirements we can ensure that the implementation checker maintain relevance for file formats such as FFV1 in AVI and FFV1 in QuickTime (both found widely in early discovery interviews and surveys). In these cases the Implementation Checker will report on the implemetnation of the supported formats specifically (here FFV1) and not focus on unsupported container formats which happen to be deployed.

Implementation Checker

The Implementation Checker, whether run via command line interface (CLI), graphical user interface (GUI), or browser-based interface, should be able to accept a file as a input. The Implementation Checker will verify that the input is relevant to the Checker (ie. is it EBML/Matroska or does it appear to contain FFV1 or LPCM). If relevant that Implementation Checker will verify adherence or document

transgressions to the test detailed in the Registry. The output will use the formats of the Reporter, specifically the PREFORMA XML format, but additional outputs in JSON, PDF, and/or HTML will be available.

Because some checks are fairly processor-intensive, MediaArea's developers will focus on the efficency of intensive checks (such as verifying the CRC payloads of FFV1 or MKV) through multi-threaded processing, frequent benchmarking, and coding optimizations. Additionally the user shall be able to express settings to balance time and thoroughness of the checks.

The graphical user interface of the implementation checker shall provide a configuration interface where sets of registrered checks may be enabled or disabled. The interface shall link each check to documentation (internal or online) regarding the context of the check. Another interface shall allow the user to select one or many files to verify against the selected checkers. The test (associatation of files to checks) may then be run or sent to a scheduler within the Shell.

The reporting of the Implementation Checker should associate checks to specific files and provide detailed context: byte offsets, links to background infromation, citations, and other supportive documentation. ## Policy Checker - Graphical User Interface

Introduction

The policy checker graphical user interface will allow set of policy rules to be created and configured to specfic workflows and then applied to selected collections of files or file-queries across a file system. Although predefined presets are available, users shall be able to use the interface to create or edit policy check presets for reuse. A policy check preset will include a list will essentially be a recipe for evaluating the technical aspects of a set of files. Although MediaArea's implementation here will focus specifically on the selected formats of Matroska, FFV1, and LPCM, the policy checker will be able to preform checks on any type of file supported by MediaInfo.

MediaArea currently manages a registry of terminology to express a diverse set of technical metadata characteristics, including information about containers, streams, contents, and file attributes. Such information is derived from container demuxing and specialized bitstream analysis. The approach of MediaInfo's design is to trace through the entire structure of a file and interpret all data in a manner that categorizes it to the content of the utilized formats underlying specifications. This interpretation of audiovisual data is comprehensive and specification based, but to supported more generalized use within media workflows, the information is then aligned to a pre-defined set of technical metadata terminology.

MediaArea will extend its matroska demultiplexer to focus on the complete potential expression allowed by Matroska 3 finalized and 1 experimental version as well as common Matroska format profiles such as webM. Additionally MediaInfo will continue work on its FFV1 bitstream filter to efficiently analyze single frame, partial streams, or whole streams. Such work will distinguish between contextual and technical metadata unique to the 4 existing versions of FFV1.

Within the use of the policy checker the user shall be able to comprehensive and conditional policy checks to apply to one or many files. The policy checker may be used to assess vendor deliverables, consistency in digitization efforts, obsolescence or quality monitoring, in order to better control and manage digital preservationc collections.

Design & Functional Requirements

Standardizing Policy Expressions To faciliate interoperability between PREFORMA's developed Policy Checkers and the Shells of all suppliers, MediaArea proposes that all suppliers collaborate to define a common data expression for policy. The policy expression standard to cover a shared registry of technical metadata that shares overlapping scope amongst all supplier policy checkers (for example: policy expressions to test against file size or file name should express those technical concepts identically). Additional policies expressions from all Suppliers should share a concept list of operators such as 'must equal', 'greater than', 'matches regex', etc. A common policy expression shall also share methods to define conditional policy checks, for instance to conditionally test if the frame size is 720x576 is the frame rate is 25 fps and to test if the frame size is 720x486 if the frame rate is 30000/1001.

MediaArea recommends the policy expressions be standardizing with an XML Schema co-defined by all Suppliers, managed with version control in a common GitHub account, and accessible through the documentation of the Open Source Portal.

Policy Designer Interface The policy designer interface enables users to create, edit, and share sets of policies. The functions of the policy designer interface shall include allowing the user to:

- create, edit, and remove policy sets
- name and describe policy sets
- import, export, and validate XML files of policy sets
- allow new policies to be added, edited, or removed from policy sets
- synchronize a vocabulary between the terminology registered in MediaInfo and the user interface
- use a list of policy test operators as defined in the policy expression XSD
- allow policies to be encapsulate within over conditional tests with an if/else approach

The policy designer interface will inform the user with a general indicator as to how time-consuming the test if. For instance to test that all selected Matroska files are version 3 or higher may be done with a quick header parse, but to test that all FFV1 frames have valid embedded CRCs would take a lot more time. Although policy sets may provide an indicator as to the amount of time required (i.e. which sets are quick and which are intensive). This information may be used to set up task scheduling efficiently.

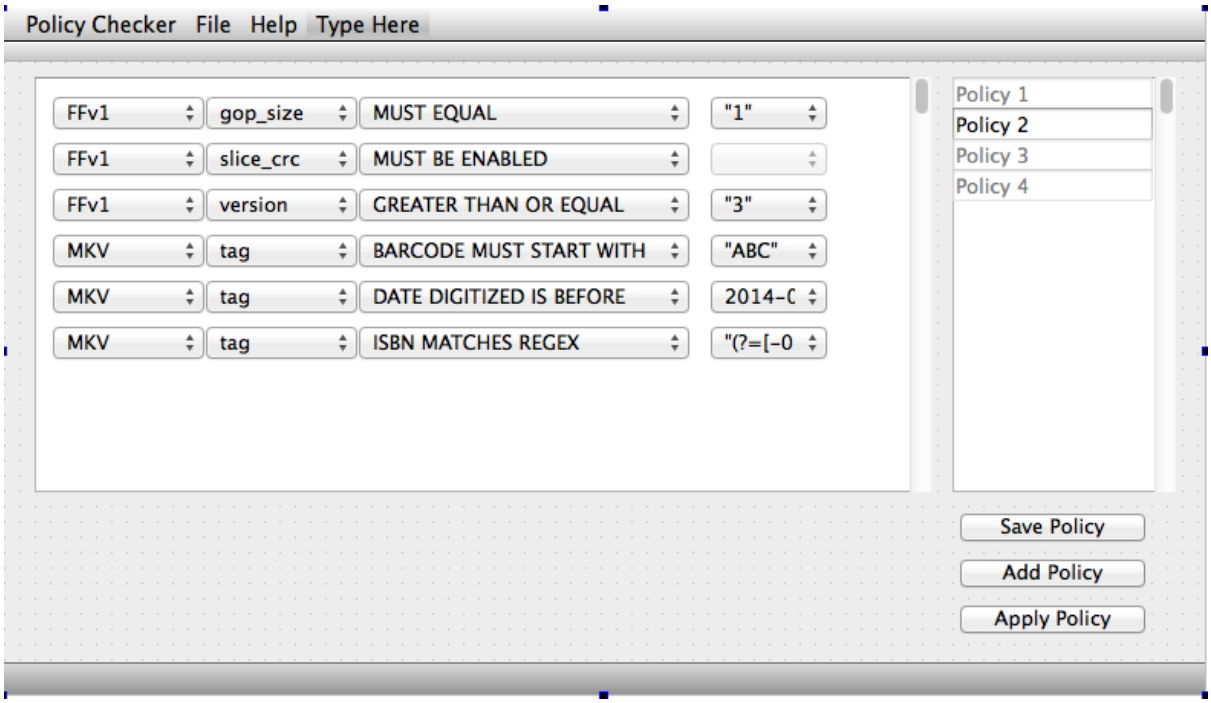


Figure 10: policy designer interface

File Selection Interface The file selection interface shall allow the operator to load files through such methods as drag-and-drop, selection through system dialogs, or file system queries. Such sets of files or queries to identify files may then be saved, name, and described. Thus an operator may define a particular directory as an entry point for acquisitions, a queue for quality control assessment, or archival storage.

Policy Test Interface Within the Policy Test Interface, the user may associate policy sets and file selections to form a policy test. Policy tests may contain name and descriptions which will be passed through to the resulting report to provide context. The policy tests may then be run directly or assigned

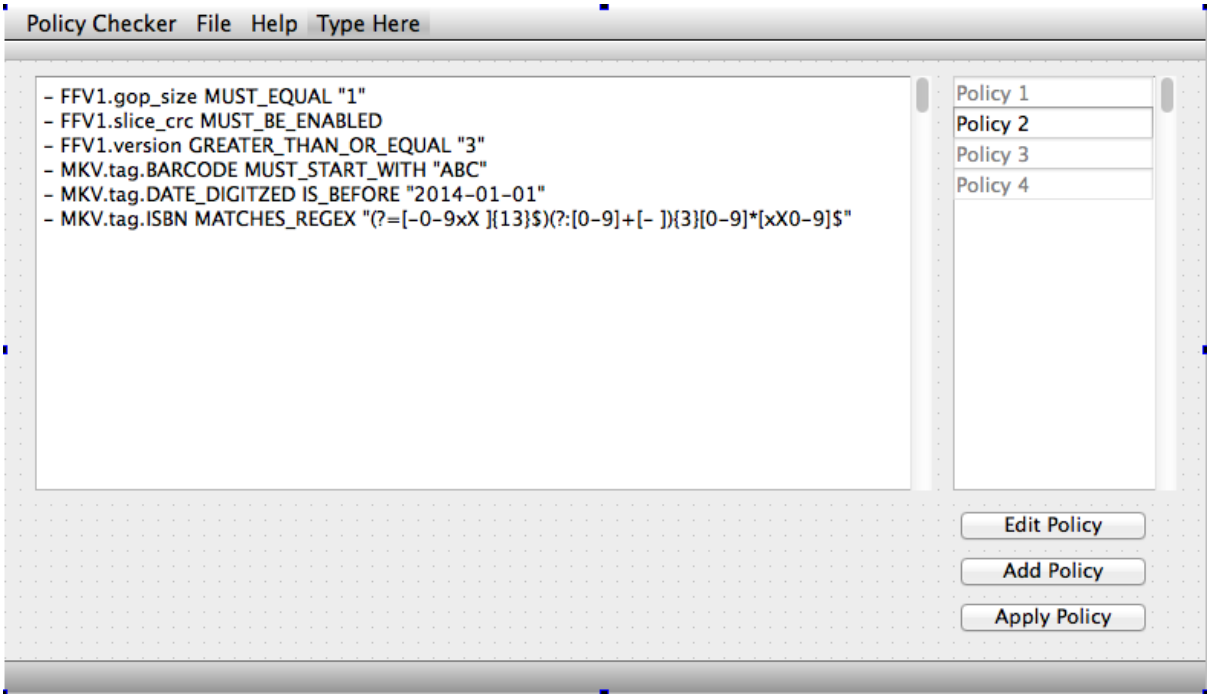


Figure 11: policy selection interface

to the task scheduler of the Shell. Within the Policy Checker GUI the results of the policy tests may then viewed or analysis within the Reporter GUI.

The policy test interface will provide access to a log of policy tests, the context of each test, and any errors in running the policy test (such as the file selection being unavailable).

Policy Checker - Command Line Interface

Functional Overview

In the context of the Command Line Interface the Policy Checker shall allow for the Policy Set XML and the file or files to be tested to be inputs to the command line utility. The output of the utility will be a report compliant with PREFORMA reporting format for the expression of conformance, policy and metadata information. The command line will provide various levels of verbosity in order to show progress in processing through multiple files and allow the policy results to be shown as they are assessed.

The Policy Checker will support exit codes to programmically inform to the output of the process.

Design and Functional Requirements

Users of the command line interface will be prompted with several usage options to call specific project APIs, including the Reporter, implementation checker, Policy Checker, and Metadata Fixer. An additional call for "–Help" will prompt an extended options menu. The functions of the command line interface shall include allowing the user to:

- create, edit, and remove policy sets
- name and describe policy sets
- import, export, and validate XML files of policy sets
- allow new policies to be added, edited, or removed from policy sets
- allow policies to be encapsulated within over conditional tests with an if/else approach

Batch Policy Checking The command line interface allows for efficient batch policy checking through calls made to the API.

Policy Checker - Web Interface

Metadata Fixer – Graphical User Interface

Introduction

Although many audiovisual formats contain comprehensive support for metadata, archivists are eventually faced with a dilemma regarding its application. On one hand, a bit-by-bit preservation of the original data comprising the object is a significant objective. On the other hand, archivists prioritize having archival objects be as self-descriptive as possible. While the OAIS model aims to mitigate such a dilemma through the creation of distinct Information Packages, this is not often the case. For an institution managing digital files as objects for preservation, a change to the file's metadata is a change to the object itself; significant attributes such as file size and checksum are irreversibly altered. Because such revisions to the object prevent fixity functions, the decision on whether or not to fix or add metadata within the OAIS structure is oftentimes complex.

The intrinsic design of the Matroska file format aims to find a balance between these two considerations. Rather than relying on an external checksumming process to validate the fixity of the file, Matroska provides a mechanism for doing so internally. The CRC elements of Matroska may be used within any Element to document the subsequent data of the parent element's payload. With this feature a Matroska file may be edited in one particular section while the other sections maintain their ability to be easily validated. Thus in addition to (or possibly in lieu of) generating a file checksum during acquisition, a archive may use PREFORMA's Matroska tools to embed CRC elements if they do not already exist. When a Matroska file is internally protected by CRCs, the sections of the file may be edited or fixed while maintaining a function to verify the un-edited functions.

Because of these Matroska features, we are very interested in how archivists may work more actively with internal file metadata through various parts of the OAIS framework. For instance reports on file edits, repairs, and outcomes of preservation events may be added directly to the file. With such tools as proposed by this project, archivists and repository systems may work with living Matroska preservation objects which internally define the context and lifecycle of themselves over time while maintaining the fixity features of the contained audiovisual data which is the essence of the overall preservation focus and what the Matroska container may be used to describe, validate, and support.

Although the Metadata Fixer can provide comprehensive levels of control over metadata creation and editing, the central objective of the metadata fixer is to facilitate repair procedures for conformance or policy issues. The Tag validation status will be presented in detail or summarization dependent on the active layout and related problematic aspects of the file with designed repair solutions. Because metadata fixes or repairs will alter a preservation file, MediaArea has dedicated a signficiant level of caution to the design of these operations. Learning more our similar experience with BWF MetaEdit, such designs will be based off a thorough programmatic understanding of the file, the actions to move the file towards a greater level of conformity, and the risks associated with doing so. The interfaces designed here will serve to intuitively relate file issues, with programatically proposed fixes, and inform to provide the user with an understand of the context and risk of the fix.

Design & Functional Requirements

File List Layout The GUI version of the metadata fixer will provide an interface to see a table of summarized metadata for one or many open files. The intent is to go allow files to be sorted by particular technical qualities or the content of embedded metadata. A table-based presentation will also allow the inconsistancies of technical metadata to be easily revealed and repaired.

MediaArea has developed such interfaces in other conformance- and metadata-focused projects such as BWF MetaEdit and QCTools and plans to use the File List Layout as an interface center for batch file metadata operations.

Customizable Sections The contents of the File List will be configurable according to the metadata values indexed by MediaInfo during a file parse. In the case of Matroska files these metadata values will also be categorized according to their enclosing Matroska section. These sections include:

- Header
- Meta Seek
- Segments
- Tracks
- Chapters
- Clusters
- Cueing Data
- Attachment
- Tagging

In additional to Matroska sections a category of file attribute data will also be provided to show information such as file size, file name, etc. Additional a 'global' section is provided to show summarization of the file's status and structure.

A toolbar in the File List Layout will enable the user to select one or many sections to allow for focus on a particular section.

As an example, checking to show the columns associated with the Matroska Header shall reveal columns such as:

- File format (Matroska, Webm, etc)
- Format version (version of Matroska, etc)
- Minimum read version

A global section would provide informational columns such as:

- Amount of VOID data with the Matroska file
- Percentage of CRC coverage with the Matroska file
- Number of metadata tags
- Number of chapters
- Number of attachments

As metadata tags may vary substantially, the tagging section of the File List Layout will show selected level 4 metadata tags as well as a column to summarize what level 4 metadata tags are unshown. Columns values which show level 4 metadata tags which contain child elements shall note visually when that tag contains child tags and reveal a summarization of child values over mouse-over. Further interaction with of metadata tags in level 5 and below can be better found in the (Metadata Editor Layout)[####mkv-metadata-editor-layout] which shall be linked from each row of the File List Layout. Within the tag setion of the of the File List Layout the shown Level 4 metadata tags may be edited directly.

The order and selection of viewed columns within the File List Layout may be saved and labelled to configure the display. This feature will allow users to design and configure layouts for particular metadata workflows. MediaArea plans to provide specific layouts in accordance with the objective of particularly OAIS functions, such as to supply contextual metadata about a digitization or acquisition event.

Managing State of Metadata Edits/Fixes A toggle within the toolbar will switch the table's editable entries from read-only to editable, to help prevent inadvertant edits. Each row of the File List Layout shall contain a visual status icon (File Edit State Icon) to depict the state of the file's metadata state. The File Edit State Icon will show if the file has been edited through the UI to different values than the file actually has; for instance, if the file must be saved before the shown changes are written back to the file. Metadata values within editable layouts shall appear in a different font, style or color depending on if they show what is actually stored or altered data that has not yet been saved back to the file. By selected a row which has an edited but unsaved state, the user shall be able to selet a toolbar option to revert the file's record back to its original saved state (to undo the unsaved edit).

Relational to Conformance / Policy Layouts The File List Layout shall contain a column to summarize conformance and policy issues with each open file and link back to the associated sections to reveal more information about these issues.

MKV Metadata Editor Layout The Metadata Editor Layout is designed to efficiency create, edit, of fix metadata on a file-by-file basis. The interface will show the contents of the Matroska tag section and provide various UI to facilitate guided metadata operations, such as providing a date and time interface to provide expresses for temporal fields, but also allowing text string expressions for all string tags as Matroska allows.

- Provide a table to show one row per metadata tag
- Provide columns with following values
  - Hierarchy
    - A relator to link tags to one another in parent/child relationships
    - A UI toggle to show or hide child metadata tags
  - Target Section
    - TargetTypeValue
    - TargetType
    - TargetSummarization of track, edition, chapter, and attachment targets
  - Metadata Section
    - TagName
    - TagLanguage
    - TagDefault (boolean)
    - TagContent (combination of TagString, TagStringFormatted, and TagBinary UI)
  - Tag Status Section
    - Tag validation status (alert on tags adherance to specification rules, logical positioning, and formatting recommendation)

Interface Notes Hierarchy

Each row of the metadata tag table may be freely dragged and dropped into a new position. Although this is usually semantically meaningless, the user should be able to organize the metadata tags into a preferred storage order. An example of this express in the UI could be that the Hierarchy column shall contain a positioner icon that the user may grab with the mouse to position the row in a different order. The positioner icon should also be able to be dragged left or right to affect the neighbor or child relationship to the metadata tag positioned above. For instance if there are two tags in the table at the same level called with the first called ARTIST and the second URL then both the ARTIST value and URL value refer to the declared target. However if the positioner icon of the URL tag row is moved to the right then the UI should indictate that the URL tag is now a child of the ARTIST tag, and thus the URL documents the URL of the ARTIST rather than the target.

Target Summarization

Each metadata tag may be associated with the context of the whole file or many specfic targets. For instance a DESCRIPTION may refer to one or many attachments or a particular chapter or a particular track, etc. In order to show the targets concisely the UI should present a coded summary to show one value that indicates the type and number of related target. The TargetSummarization may how "A3" to indicate that it refers to the third attachment, or "T4" to indicate that it refers to the fourth track. When the TargetSummarization is moused over a popup should reveal a list of associated targets with the UID and perteninet details of each target as well as a link to jump to a focus of that target within its corresponding layout (such as the Chapter Layout or Attachment Layout).

Tag Content Bahavior

Matroska tags may contain either a TagString or TagBinary element. When single-clicking or tabbing into a TagContent field then if the TagContent is a TagString it shall be directly editable and if the

TagContent is a TagBinary the TagContentModalWindow shall appear selected to the Binary tab with a guided hex editor.

Tag Content Modal Window

The TagContent Modal Window is a UI designed to accommodate editing of TagContent or TagBinary values. The UI shall contain three tabs:

- String Editor
- Formatted String Editor
- Hexadecimal Editor

When creating a new metadata tag in a matroska file. If the TagName corresponds to a binary type it will open the Tag Content Modal Window to allow to the binary data to be provided, else it will default to allowing the metadata tag value to be edited within the string box of the layout in which the metadata tag was created. If the tag name of the newly created metadata tag corresponds to a binary type then the Hex Editor tab of the Tag Content Modal Window will be used.

The Hex Editor tab of the Tag Content Modal Window will allow for hexadecimal editing, allow data to be loaded to TagBinary from a selected file, or saved out to a new file.

When doubling clicking on an existing metadata tag in an editing or file list layout the Tag Content Modal Window shall open to reveal the most appropriate editing tab. If TagBinary is used then the Modal Window shall open to the Hex Editor tab. If TagString is used than it should use the Formatted String Editor tab if the data complies with the formatting rules, else use the String Editor.

Validation Status

The validation status indicators and associated procedures are central to the objectives of the Metadata Fixer. The Matroska specifications is rich with precise formatting rules and recommendations that are intended to facilitate the predictability and inter-operability of the file format; however, many Matroska tools and workflows make it easy to inadvertantly violate the specifications or cause conformance issues. The Metadata Fixer layouts will provide a visual indicator of validation status issues, so that when files are opened any validation issues are clearly show in relation to the invalid section and linked to appropriate documentation to contextual the issue. Additionally is an operator makes a modification that is consider a validation issues, the user will be informed to this issue during the edit and before the save. If the users tries to save metadata edits back to a Matroska file while their metadata edit contains validation issues, the user must confirm that this is intended and that the result will be invalid.

In many cases repairs to well defined validation issues are repairable programmatically. The Validation Status section of layouts will show related repair procedures (if defined) and summarize (to the extent feasible) the before-and-after effects on the file.

Metadata Import / Export Both the GUI and CLI of the Metadata fixer will allow Matroska metadata tags to be imported into or exported from a Matroska file using Matroska existing XML tagging form. In addition to information typically found in Matoroska's XML tag format, information on validation status will be included.

Layout Preferences

- Checkboxes to disable appearance of columns in File List Layout
- Functions to allow the currently selected File List Layout options to be saved and labelled
- An ability to load pre-designed or user-created File List Layout options
- A list to specify level 4 Matroska tags to appear in File List Layout as a column
- Default value to use for the default value of TagLanguage on new metadata tags

Metadata Fixer – Command Line Interface

Functional Overview

The Matroska Metadata Fixer command line interface will provide repository systems with a means to automatically assess what potential fixes may be performed, selectively perform them, add or changes files metadata, or preform structural changes to the file.

Overall all the features documented in the Metadata Fixer GUI are also feasible within the CLI, although some scripting may be necessary around the CLI to emulate a fully programmatic performance of all anticipated GUI workflows.

Functional Requirements include:

- accept one or many Matorska files as well as one or many PREFORMA policy speciications (via xml) as an input
- generate a text based representation of the EBML structure in json or xml, which identifies and categories EBML sections and which includes attributes to associate sections of the EBML structure (or the file itself) with registered conformance or policy errors
- validate a Matroska file against conformance or policy errors and generate a text based output which summarizes errors with associated fixes
- preform and log identified metadata fixes
- add, replace, or remove Matroska metadata values based on a developed EBML equivalent of XPath

Metadata Fixer – Web Interface

Reporter

Functional Overview

The Reporter portion of the conformance checker presents human and machine-readable information related to implemenation and policy checks, metadata fixes, check statistics, conformance priorities, and other associated session documentation. This information is derived from the database and multiple component project APIs, passed through the PREFORMA core and finally combined and transformed into a desired output. Several output formats allow for external software agents to futher process this reportage.

The Reporter may accept a previously-generated PREFORMAXML or other supported output format for collation with other conformance checks. Previously-generated PREFORMAXML reports may also be transformedn into additional desired outputs.

Design and Functional Requirements

- human-readable supported output formats: PDF, TXT
- machine-readable supported output formats: PREFORMAXML, JSON
- optional machine-readable supported output formats: CSV/TSV
- report on implementation checks, policy checks, and metadata fixes
- report on information concerning preventative measures for non-conformed files
- option to report verbose bit traces of individual files
- batch reporting features the option to nest specific objects.

Contents

Report Name (“eg.,”Conch Report“), XXXX-XX-XX XX:XX:XX Date (ISO spec)

Implementation Checking Errors Implementation Chcking Warnings

Policy Checking Errors Policy Checking Warnings

Metadata Fixing Documentation (Action measures and preventative measures for non-conformed files)

Individual File General / Verbose bit metadata readouts

Reporter - Graphical User Interface

The Reporter's Graphical User Interface displays human and machine-readable data to the end user. Through a drop-down menu, a user can select the above mentioned output handlers.

Reporter - Command Line Interface

On the command line interface, a user would be able to export a report using flags designating output and output format (e.g., "–output=XML").

Reporter - Web User Interface

Like the Graphical User Interface, the Reporter's Web User Interface displays human and machine-readable data to the end user. Through a drop-down menu, a user can select the above mentioned output handlers. # Style Guide

Source Code Guide

Portability

Excelling in cross-platform open source development, MediaArea will utilize tools throughout the development phase in order to provide users with a downloadable source code that offers functional portability between the different deployment platforms (MS Windows 7, Mac OSX, and Linux).

The source code can be shared and used between the targeted platforms and will run and behave similarly across different users' machines. The various releases will be implemented with adaptable interoperability between these platforms and the software will run dependent on which downloadable source code and executable the user's platform requires. This means MediaArea's open source project, throughout the development and deployment phases, will continue to always provide downloadable access to software and support for each of these individual platforms.

Modularity

MediaArea's regularly released source codes will be developed within a modularized architecture in order to create an unrestricted atmosphere for the improvement of the project's maintainability and assembly. Conforming to this development technique and creating distinct, interchangeable modules will allow the software, and its corresponding open source community, to remain sustainable in its growth and facilitate ongoing collaborative feedback throughout the development phase. Each module will have a documented interface (API) that defines its function and interactive nature within the software.

The construct and eventual structure of modularity within this project will be key to the health and sustainability of the project's potential success as a fully integrated implementation checker. The regular release of source code for this project, built within this architecture, will enable better feedback and issue tracking from both users and memory institutions utilizing the software.

Deployment

MediaArea will develop a implementation checker that is designed to allow for deployment in the five following types of infrastructures and environments:

- PREFORMA's website
- Within an evaluation framework
- Within a stand-alone system (MS Windows 7, Mac OSX, and Linux)
- Within a network-based system (server or cloud)
- Within various legacy systems

Access release for the intended targeted users will be supported within all of these environments. In order to demonstrate the project's successful deployment within these infrastructures, MediaArea will, in addition to supplying the standard corresponding technical documentation for each, undertake the following considerations:

The project's necessary PREFORMA website, including centralized links to all open source materials and community outreach, will be considered as the official deliverable for the entire project. The associated and required documentation, tools and instructional feedback will also be provided and accessible on the PREFORMA project website as well as the open source platform.

The implementation checker will also fulfill the requirement of being deployed within the direct infrastructure to facilitate evaluation and use within the PCP system.

For stand-alone users or smaller institutions, the implementation checker will be fitted with the capability to be packaged, downloaded and run as an executable on machine's running any form of a standard operating system (MS Windows 7, Mac OSX, and Linux).

The implementation checker will be developed to deploy within different network-based solutions and environments (including dedicated servers and cloud solutions) hosting the memory institutions' digital repositories.

Via written API integration, the implementation checker will also be able to properly function when plugged into various legacy systems.

APIs

Via APIs, the developed and deployed implementation checker will be designed to interface and integrate with other software systems. Programming tools and software standards and practices will be upheld in order to allow for a potential software-to-software interface. A long-term sustainable usage and presence within the open source community will be further enhanced via correct implementation of this successful API integration.

The outcome of the ability to interface with other software systems is that the software and technical documentation will have ongoing support and integration within the individual workflows of the memory institutions' preservation plans.

Open Source Practices

Development

MediaArea's open source software development within the PREFORMA project will establish and uphold open source work practices and standards. These practices will abide by the following project rules:

- Use of nightly builds: A nightly build is an automated build that reflects the most up-to-date version of developed software's source code. Users will have access to these nightly builds as their release will allow for collaborative groups of developers and users to work together and continuously gain immediate feedback and fixes to the most current state of the software. With access to the absolute latest versions, MediaArea and all open source collaborators will more readily gain insight

into potential bugs and issues that could arise during the development phase. Programmers will be able to determine if they "broke the build", making the software inoperable with their latest code. Immediate access to fixing these issues can be made more efficiently.

- Use of software configuration management systems: Operating with a software configuration management system (Git) will allow MediaArea easy version control as well as knowledge of the revisions needed. This is an essential part of the open source community that allows developers to be able to work together collaboratively. A version control system allows multiple people to work on the same or similar sections of the source code base, simultaneously and at the same time, with awareness and prevention of overlapping or conflicting work. Git will be used as the software configuration management system for this project.
- Use of an open platform for open development: MediaArea will operate within an open source platform on which to develop software and better facilitate the open development of that software. Public visibility that allows anyone the ability to contribute to the software's development allows for sturdier, more reliable outcomes. Feedback is more easily sought and more readily provided with the use of an open platform. Github will be used as the open platform for open development of this project.

Open Source Platforms

All software development as well as the development of all relevant and corresponding digital assets and tools created by MediaArea during the PREFORMA project will exist and function as an open source project within an open source platform (Github). This open source development platform will offer full transparency and traceability throughout the development phase and facilitate a functional collaborative environment with developers, users, stakeholders, and institutions.

Source code, issue tracking, documentation, updates, release and various forms and channels of public outreach will be centralized within the open development platform and linked to within the PREFORMA project page.

Contribution Guide

File Naming Conventions

All project files related to documentation regarding the PREFORMA project will be named using CamelCase. Project documentation's actual sample data will be shared using the snake_case. These objects should carry a suitably descriptive file name that elaborates on the contents of the file and follow the standard practices and expectations of their corresponding naming conventions and specifications. File naming conventions and rules will be upheld in order to implement an efficient database of document and file releases within the open source community. In regards to the required conventions for commit messages on the open source platform, all messages should be concise and clear and effectively summarize each contribution to the project. If more than one change was made, users should not create one commit message to cover all feedback and changes. New individual commits should be made to cover each individual change made to the relevant file being altered. Effective commit messages, covering context of a change, will enable MediaArea to work within a speedier, more efficient review process and better alter development around this feedback.

Rules for Qt/C++ code

MediaArea's open source project will be programmed in C++ and will use the Qt application framework.

Guideline for Qt is as follows:

MediaArea will follow the applicable rules for programming within the Qt cross-platform application development framework.

Attention to detail will be given to the following rules/guidelines:

Indentation: - Four spaces to should be given for indentation (not tabs)

Variables: - Each variable should be declared on separate lines, only at the moment they are needed - Avoid short names, abbreviations and single character names (only used for counters and temporaries) - Follow the case conventions for naming

Whitespaces: - Use only one blank line and use when grouping statements as suited. Do not put multiple statements onto one line. - Also use a new line for the body of a control flow statement - Follow the specific single space conventions when needed

Braces: - Attached braces should be used (follow guidelines for rules and exceptions) - Curly braces are used only when the body of a conditional statement contains 1+ line or when body of a conditional statement is empty (follow guidelines for rules and exceptions)

Parenthesis: - Parenthesis should be used to group expressions

Switch Statements and Jump Statements: - Case labels are in the same column as the switch - Each case should have a break statement at the end or a comment to indicate there is no intentional break - Do not use 'else' at the end of Jump Statements unless for symmetry purposes

Line Breaks: - Lines should kept under 100 characters - Wrap text if necessary - Use commas at the end of wrapped text and operators at the beginning of new lines

Exceptions: - Always try to achieve functional, consistent and readable code. When code does not look good, exceptions to the rules may pertain to fixing this situation.

For more specific rules, examples, exceptions and guidelines, please refer to the Qt Coding Style guide: http://qt-project.org/wiki/Qt_Coding_Style

Guidelines for C++ code is as follows:

Manageability and productivity within the C++ coding atmosphere will be preserved by upholding to the Style and Consistency rules necessary for creating a readable and controlled code base. Attention to detail will be given to the rules governing the creation of a workable open source code in the following areas:

- Headers
- Scoping
- Classes
- Naming
- Comments
- Formatting
- Specific Features/Abilities of C++
- Relevant Exceptions

For a detailed account of specific rules, examples and guidelines for each section, please refer to the Google guide on C++: http://google-styleguide.googlecode.com/svn/trunk/cppguide.html

Rules for contributing code

Contributions of code or additions to MediaArea's PREFORMA project documentation must be written with Qt (following the advised standards and practices) and must be made in the form of a branch submitted as a pull request.

- Create your feature branch (git checkout -b my-new-feature)
- Commit your changes (git commit -am 'adds some feature')
- Push to the branch (git push origin my-new-feature)
- Create a new Pull Request with a more verbose description of the proposed changes

Link to github repository: (https://github.com/MediaArea/PREFORMAMediaInfo/fork)

Rules for contributing feedback

Feedback of all kind is encouraged and can either be made through opening an issue on Github or by contacting the team directly at info@mediaarea.net.

Issue tracking and feedback will be encouraged directly through the open source platform (Github) around which, in addition to PREFORMA's Open Source Portal, MediaArea will function and centralize the anticipated infrastructure for a collaborative community environments. In addition, contributions and feedback can be left via either the IRC channel or the mailing lists pertaining to the project.

Linking

MediaArea will implement linking throughout the open source community in order to create a sustainable and documented infrastructure that facilitates clarity in the progression of the project. In order to produce an environment that offers both the users and MediaArea a space for descriptive feedback, intuitive discoveries within the code, and the ability to resolve issues, linking will function through the released source code, the corresponding software documentation, a ticketing system, general feedback, and potential commit messages. In one such example, as the registry itemizes user's individual conformance checks and tests, these tests will subsequently link to code blocks and commits so that the software can continue to be developed and associated to that conformance check.

If this habit is implemented efficiently, MediaArea will create an open source community that enables ease in interacting and reviewing with both user-friendly and human-readable descriptions of conformity checks combined with their related programmatic results.

Test Files

MediaArea's test files and media will exist in the SampleTestFiles folder within the open source platform. This designated sample folder will also be broken down into separate folders for each relevant file format and the separate specification parameters set for testing.

It is anticipated that a large library of reference media and test files will be created to highlight the different outcomes associated with issues and errors that may arise in regards to certain files and specifications push through the software. The test files will either be self-created, solicited, or pulled from a variety of online reference libraries.

This curated selection of tests will include the following:

- files that conform to the relevant file format's technical specifications
- files that do not conform and therefore deliberately deviate from the file format's technical specifications (in association with the appropriately coded error messages)
- specific files that originate from and/or adhere to the technical specifications of the file formats from participating memory institutions (including examples that both conform to and deviate from the requirements)

Because it is crucial to the stimulation of a sustained and well documented open sourced community, the resulting issues and feedback from the testing of these created and solicited files will also be documented and will contain information on the relevant version of software used for the test.

Release Schedule

MediaArea intends to release various versions of all relevant source codes and executables for each of the deployment platforms that the project will be configured to perform upon successfully. For stable versions of the software, new downloads and rolling releases will be provided and made available on a monthly basis. Stable versions will take into account software fixes, updates, and bug reports throughout the development phase and additionally will have gone through a QA process during that time.

Certain deployed (LTS) versions, upholding the build of the stable versions, will be provided and released during the required delivery stages of the PREFORMA project and will be developed as sustainable for a long period of time within the open source project.

New nightly builds and updates of the source code will also be made available to download during all stages. This ensures that all users and organizations will have access to downloading the most up-to-date version of code that exists throughout the project.

Downloads will be made available through a public repository with a functioning issue tracker (Github). In conjunction with the releases, a roadmap will be created in order to track these updates publically and encourage open collaborative usage and issue feedback. Both the older and more recent development, stable, and deployed (LTS) versions will be made available to users of any level, throughout these multiple platforms, for the entirety of the project. If a user wishes to download an older version of the source code or executable, MediaArea will have this option available.

All source codes and updates will be made accessible on the following platforms:

- MS Windows
- Mac OSX
- Linux (Ubuntu, Fedora, Debian, and Suse)

In regards to the nightly source code builds and monthly stable version releases, MediaArea will facilitate easy access for users to download each of these different versions with the creation and upkeep of a single file that contains all of the necessary open source tools. Taking into account the varying deployment platforms, a different file (containing all relevant documents) will be created for each. The expected and differing standard procedures and patterns between the different platforms, as well as their individual configurability rules, will be upheld throughout the development and release phases. Support will be accessible via all of these platforms.

Along with the downloadable codes and tools, full supplementary documentation, developed to suit users functioning on each platform, will be included within the release schedule and will stay up-to-date with those releases. The necessary steps required in downloading and extracting MediaArea's source code and software build, in order to created a directly executable object on the user's machine, will be fully documented for every type of user. We intend on establishing and releasing informational documentation including detailed, step-by-step instructions for both a non-technical and highly technical user or institution.

License

All software releases and digital assets delivered by MediaArea will be produced and made available under the following Intellectual Property Rights (IPR) conditions:

- All software developed by MediaArea during the PREFORMA project will be provided under the following two open source licenses:
  - GNU General Public License 3.0 (GPLv3 or later)
  - Mozilla Public License (MPLv2 or later)
- All source code for all software developed by MediaArea during the PREFORMA project will always be identical and functional between these two specific open source licenses ("GPLv3 or later" and "MPLv2 or later").
- All open source digital assets for the software developed by MediaArea during the PREFORMA project will be made available under the open access license: Creative Commons license attribution – Sharealike 4.0 International (CC BY-SA v4.0). All assets will exist in open file formats within an open platform (an open standard as defined in the European Interoperability Framework for Pan-European eGovernment Service (version 1.0 2004)). ##Conclusion

The PreForma project challenge presents an opportunity for stakeholders of digital preservation to work together and develop meaningful solutions to file format conformance issues. MediaArea is proud to be among the several successful applicants of the initial PreForma tender, and is confident that our submitted Phase 1 proposal for the creation of a Matroska, FFV1 and LPCM conformance checking toolset will greatly enhance the project's overall scope.

At the core of the PreForma project is the conformance checker, which, when produced, will implement meaningful file format validation for long-term digital preservation. MediaArea is focused on developing the checker and its related environment. MediaArea's previous work MediaInfo and QCTools projects have led to a refined project development with risk management strategies.

While LPCM endures as a prevalent raw audio stream, Matroska and FFV1 remain largely as outliers of digital preservation policy discussions among memory institutions. We believe that this is mostly due to misapprehensions of Matroska and FFV1. In addition to the development of a conformance checker, MediaArea presents a plan to strengthen the disclosure, transparency, and credibility of Matroska and FFV1 through standardization and adoption by relevant standards bodies.

The Preforma MediaInfo team welcomes the opportunity to see the project through its completion.