PreForma MediaInfo

Technical and Architecturual Report

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:

Date: February 28, 2015

Summary:

Introduction to Architecture

This is a roadmap for the technical components of the project.

Global Architecture

The global architecture schema defines the context in which the PreFormaMediaInfo software is situated and gives a high-level understanding of the software.

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
- Conformance checker and metadata grabbing module
- Policy Checker
- Reporter
- User Interface(s)

** CLI (Command line interface)

** GUI (Graphical user interface)

** Web (Web based interface)

Checker Architecture

The Checker Architecture details the structural components of the conformance checker and metadata-grabbing module.

- Transport interface
- Container/wrapper implementation
- Container/wrapper demuxer
- Stream/essence implementation
- Stream/essence decoder
- Stream/container coherency check
- Baseband analyzer # PreForma Global Architecture

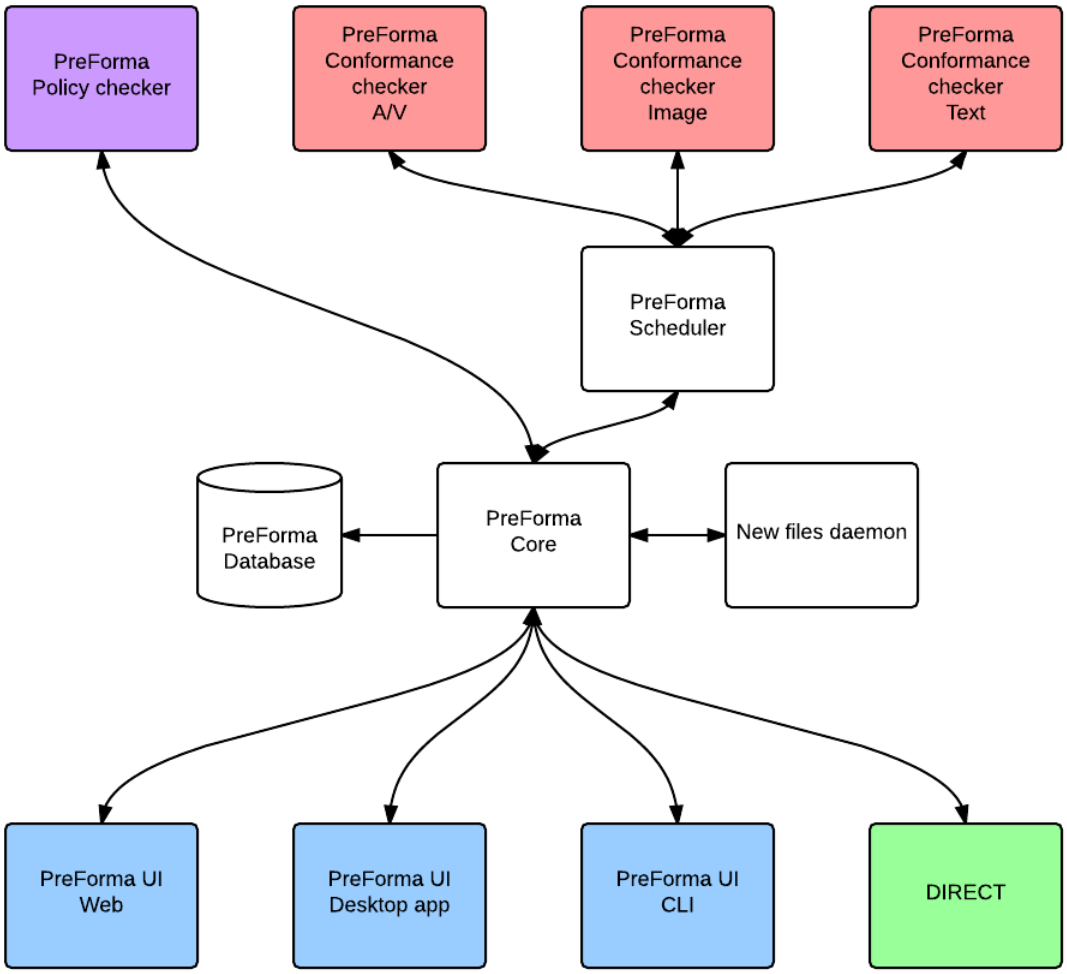

Figure 1: Global Architecture Schema

Architecture schema

Common to all elements

All elements can be installed on the same server or on different servers, depending on the expected workload.

Core (Controller)

The Core serves as communication between all plugins within and outside of the PreForma MediaInfo system and between all layers. The Core is the main service and runs in a passive, background mode.

The Core has several major functions:

- controls the checkers and manages data for the User Interface
- waits for commands from the New Files Daemon and User Interface
- send commands to the scheduler for file-checking
- 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 / 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 conformance checker, including the policy checker rules and the user rights management.

- store metadata and results of the conformance checker
- store the policy checker rules
- user rights management

Interface :

- Core : native driver

Software :

Potential database management system options, contingent on 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 conformance checkers by using a message broker interface. It translates the file data into one unified language for access within all aspects of the software.

- distributes files
- translates file data into unified language

Interface :

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

Software :

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

New Files Daemon

The New Files Daemon is a background process that listen for new files available for validating. It uses the inotify notification system API for a Linux kernel or kqueue/kevent for a BSD kernel.

Programming language : C++

Conformance Checker and Metadata Grabbing Module

This module can utilize one or more checkers for each media type. As the conformance checker process could be very long, we use an asynchronous system based on messaging system to not lock the system. Metadata and conformance check result are send back to the Core to be stored in the database.

- runs the conformance tests for the different type 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 run tests on metadata grabbed by the conformance checker and metadata grabbing module (the Checker).

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

Interface :

- Core : REST API

Programming language : C++

Reporter

Within the user interfaces are ways to export raw metadata and human-readible JSON/XML.

- 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 conformance checkers in the PREFORMA ecosystem, which allows the reported module to combine output from multiple checker components in one report. 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, and addressing improvements in the creation/digitisation process

Interface :

- Core : REST API

Programming language :

- CLI : C++
- GUI : C++ / Qt (LGPLv3+)
- Web : PHP/Symfony (MIT)

User interface

- displays test results and control the Core
- allows metadata (descriptive and structural) to be edited

PreForma MediaInfo will provide three different options for a human interface for maximum flexibility. These three interfaces are:

- CLI (Command line interface)

  A command line interface will be functional on nearly all kinds of operating systems, including those with very little graphical interface support. It allows for integration into a batch-mode processing workflow for analyzing files at scale.

- GUI (Graphical user interface)

  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.

- Web UI (server/client)

  The web interface will provide access to conformance checks without software installation.

Interface :

- Core : REST API

Programming language :

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

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

- Transport Layer
- Container Implementation (Conformance Check)
- Container Demuxer
- Stream Implementation (Conformance Check)
- Stream Decoding (through plugin)
- Stream/Container Coherency Check
- Baseband Analysis (through plugin)

Transport layer

Preforma MediaInfo: File on disk or direct memory mapping

Preforma MediaInfo natively offers a 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.

Plugin integration proof of concept: libcURL

libcURL is licensed under an MIT license that is compatible with both GPLv3+ and MPLv2+. curl offers extensive support for transferring data through many protocols. By incorporating curl into PreForma MediaInfo 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

Preforma MediaInfo: 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.

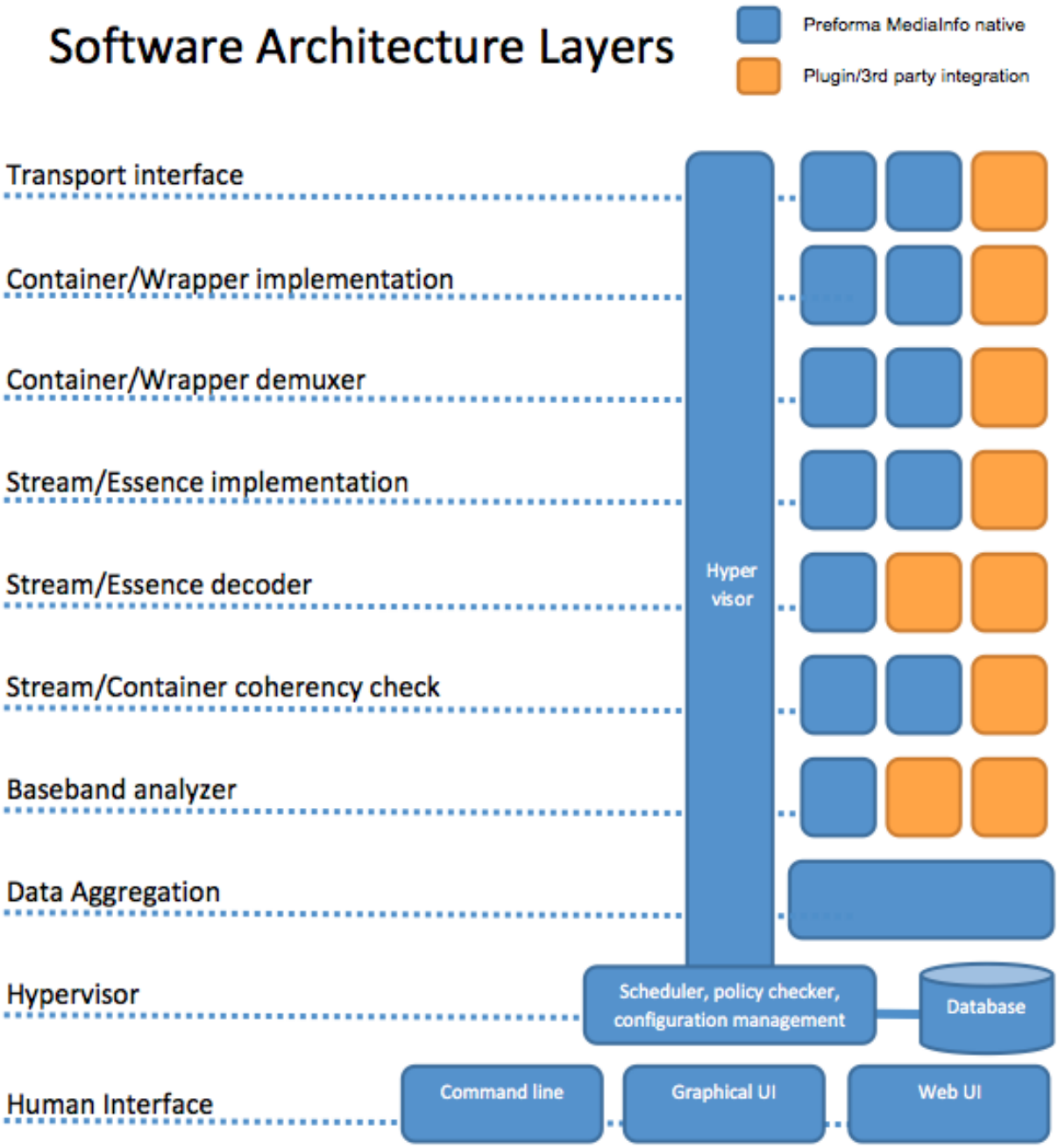

Figure 2: software architecture layers

Container/Wrapper Demultiplexing

Preforma MediaInfo

PreForma MediaInfo will utilize MediaInfo's existing demuxing libraries 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, PreForma MediaInfo 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)

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. 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.

Although PreForma MediaInfo 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 PreForma MediaInfo could separately download FFmpeg and link the two together to enable additional tools such as:

- 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.

Stream/Essence implementation checker

Preforma MediaInfo:

- 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

- 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

Preforma MediaInfo

- PCM (including D-10 Audio, AES3)

Plugin integration proof of concept: FFmpeg

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

Plugin integration proof of concept: OpenJPEG

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

Container/Wrapper vs Stream/Essence Coherency Check

Preforma MediaInfo

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

Baseband Analyzer

Preforma MediaInfo

- 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 conformance 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 conformance 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 conformance 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 conformance 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 PreForma MediaInfo 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.) ## Metadata Fixer GUI

Introduction

Although many audiovisual formats contain comprehensive support for metadata, archivists are eventually faced with a dilemma regarding it's 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 tot 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 CLI

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 # Style Guide

Source Code Guide

Portability

Source code MUST be built for portability between technical deployment platforms.

Modularity

Source code MUST be built in a modular fashion for improved maintainability.

Deployment

The Conformance Checker MUST allow for deployment in these five infrastructures/environments: The PREFORMA website, an evaluation framework, a stand-alone system, a network-based system, and legacy systems.

To sufficiently demonstrate the scope and functionality of the Conformance Checker, it, along with associated documentation and guidelines, must be made available at the PREFORMA project website. The PREFORMA website will be considered as the deliverable for the PREFORMA project.

In order to gather sufficient structured feedback on the conformance checking process, the Conformance Checker will require deployment within the DIRECT infrastructure for test and evaluation of the tool in the PCP procedure.

The Conformance Checker must have the capability to be packaged and run as an executible on a PC running any standard operating system (at least for: MS Windows 7, Mac OSX, common Linux distributions such as Ubuntu, Fedora, Debian, and Suse). This ensures the conformance checker can be used in small-scale institutions without centralized IT infrastructure.

The Conformance Checker must allow for deployment in network-based solutions (dedicated server, cloud solutions) for digital repositories.

The Conformance Checker must have the capability of being plugged into legacy systems via written API integration.

API's

The Conformance Checker MUST interface with other software systems via API's.

Open Source Practices

Development

Development of software in open source projects in PREFORMA MUST utilise effective open source work practices. Effective open source work practices include:

- use of nightly builds Nightly builds are automated neutral builds that reflect the current, most up-to-date status of a piece of developed software. Access made to nightly builds allow for groups of developers to work collaboratively and always assess the most current state of the software, with consideration to

potential bugs or other hazards that could occur during the development process. Programmers are able to confidently determine if they "broke the build" (made the software inoperable) with their code and prevent or correct changes quickly, as needed.

- use of software configuration management systems (e.g. Git) Using a software configuration management system allow for version and revision control, an essential component to developers working collaboratively. A version control system allows multiple people to work on same or similar sections of the source code base 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 (e.g. Github) An open platform on which to develop software facilitates the open development of that software. Public visibility and ability to contribute to the software by anyone allows for heartier, more reliable software. 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 development of software and all development of digital assets (related to developed open source software) in PREFORMA MUST be conducted and provided in open source projects at open development platforms.

Contribution Guide

File Naming Conventions

Files related to documentation should be named in CamelCase. Sample data should be added in snake_case with a sufficiently descriptive title.

Commit messages should concisely summarize the contribution. Commits should be cohesive and only include changes to relevant files (e.g. do not fix a typo in the Style Guide, change scope paramaters, and fix a bug all in the same commit).

Rules for Qt/C++ code:

4 spaces are used for indentation. Tabs are never used.

For more guidelines, refer to the Qt Coding Style guide: http://qt-project.org/wiki/Qt_Coding_Style For even more guidelines, Google guide on C++: http://google-styleguide.googlecode.com/svn/trunk/cppguide.html

Rules for contributing code

Contributions of code or additions to documentation must be written with Qt and must be made in the form of a branch submitted as a pull request.

1. Fork this repository (https://github.com/MediaArea/PreFormaMediaInfo/fork)
2. Create your feature branch (`git checkout -b my-new-feature`)
3. Commit your changes (`git commit -am 'Added some feature'`)
4. Push to the branch (`git push origin my-new-feature`)
5. Create a new Pull Request with a more verbose description of the proposed changes

Rules for contributing feedback

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

Linking

In order to facilitate self-description, intuitive discovery, and use of resulting code and documentation it is highly encouraged to utilize linking through documentation, tickets, commit messages, and within the code. For instance the registry itemizes individual conformance checks should link to code blocks and/or commits as software is developed that is associated to that conformance check. In this manner it should be feasible to easily review both human-readable descriptions of conformity checks and associated programmatic implementations.

Test Files

Test files exist in the SampleTestFiles folder, separated into respective folders by file format and testing parameters.

The test files include the following: * files that conform to the technical specification of the file format * files for the open source project that by design deliberately deviate from the technical specification of the file format (with proposed associated error messages) * files from memory institutions for the open source project that conform (or do not conform) to the technical specification of the file format

Release Schedule

This project is expected to release a new development version of all source code each night to be available for download in a single file for each deployment platform. This implies that any user will be able to download and compile an up-to-date version of all source code for deployment platforms relevant for PREFORMA.

There shall always be source code available for download (provided in a single file) for several different deployment platforms (at least for: MS Windows 7, Mac OSX, common Linux distributions including Ubuntu, Fedora, Debian, and Suse). For each platform specific source code version (development version, stable version, and deployed (LTS) version) there shall always be an up-to-date corresponding executable that can be downloaded as a single file. In addition, all old versions (including nightly versions) of source code and corresponding executables shall be available for download over time.

This implies that for each deployment platform (at least for: MS Windows 7, Mac OSX, common Linux distributions including Ubuntu, Fedora, Debian, and Suse) the open source project shall provide a file which contains all necessary open source tools that can be used (without access to the Internet) to create an executable for that platform. This means that at least six such files (containing open source tools) shall be available for download from the open source project. > For example, a user wishing to create of an executable for MS Windows 7 from the source code provided by the open source project on the open platform will (in principle) undertake the following steps:

```
* The complete source code for MS Windows 7 is downloaded as a single file (e.g. "source-mswin7.zip") fi
* The complete build environment for MS Windows 7 is downloaded as a single file (e.g. "buildenvironmen
* The build enviroment is used on the local machine (which now is not connected to the Internet) to crea
```

Further, as part of the content of the complete source code for each specific platform, detailed instructions for how to use the build environment to create the executable from the downloaded source code. The instructions must be provided for both technical and non-technical users, and there must be easy to follow detailed step-by-step instructions.

In addition, separate detailed instructions for how to build executables in which source code from different open source projects are integrated into one single executable must also be provided (see further section 2.4)

The source code will contain at minimum:

- Complete source code for MS Windows 7

- Complete source code for Mac OSX
- Complete source code for Ubuntu Linux
- Complete source code for Fedora Linux
- Complete source code for Debian Linux
- Complete source code for Suse Linux

The source code will be fully executible with text, images, and audiovisual files.

License

The software and digital assets delivered by tenderer are made available under the following IPR conditions:

All software developed during the PREFORMA project MUST be provided under the two specific open source licenses: “GPLv3 or later” and “MPLv2 or later”.

All source code for all software developed during the PREFORMA project MUST always be identical between the two specific open source licenses (“GPLv3 or later” and “MPLv2 or later”).

All digital assets developed during the PREFORMA project MUST be provided under the open access license: Creative Commons CC-BY v4.0 and in open file formats, i.e. an open standard as defined in the European Interoperability Framework for Pan-European eGovernment Service (version 1.0 2004)