The Definitive PDF/A Conformance Checker

PREFORMA Phase 1

Final Report of the veraPDF Consortium

Introduction

Referring to the original veraPDF Tender Proposal section 1.1, *Proposed Solution,* the mission of the veraPDF Consortium is to develop “the definitive, industry-approved, open-source implementation checker for validating PDF/A-1, PDF/A-2, and PDF/A-3.”

This document is the final report of the veraPDF Consortium on Phase 1 of PREFORMA. It includes all deliverables: plans for Community Engagement, Functional and Technical Specifications for the PDF/A Conformance Checker, and supporting documents.

Table of Contents

Glossary of Terms

Term	Definition
Assertion	Generally an Assertion is a boolean expression, i.e. it may be evaluated to true or false, used in software testing. Evaluating the Assertion involves examining some property of the item under test.
Byte Sequence	A binary data stream read from a source, for example: ● an input stream from a file on storage device accessed through a file system; ● an input stream read from a particular URL; or ● an input stream read directly from memory. A particular byte sequence is identified by combining its length in bytes and the SHA-1 Hash (derived from its contents).
Conformance Checker	A PREFORMA term that defines a generalised open source toolset, i.e. not concerned with a particular file format, that: ● validates whether a file has been produced according to the specifications of a standard file format; ● validates whether a file matches the acceptance criteria for long-term preservation by the memory institution; ● reports in human and machine readable format which properties deviate from the standard specification and acceptance criteria; ● performs automated fixes for simple deviations in the metadata of the preservation file.
Embedded Resource	A Byte Sequence embedded into the PDF Document such as an image, font, colour profile, or attachment.
Embedded Resource Parser	A third-party tool which can parse and analyse Embedded Resources, for example a JPEG2000 validator or font validator.
Embedded Resource Report	A Machine-readable Report produced by an Embedded Resource Parser containing information about an Embedded Resource.
Human-readable Report	A report generated from a Machine-readable Report in a format suitable for human interpretation (eg. HTML or PDF) and containing messages translated according to a Language Pack.
Implementation Check	The execution of a discrete Validation Test for a particular PDF Document. The Implementation Checker carries out these tests when validating a PDF Document against an ISO standard.
Implementation Checker	A PREFORMA term for the component which “performs a comprehensive check of the standard specifications listed in the standard document.” See PDF/A Validation.
ISO Working Group (WG)	An ISO committee for one or more ISO standards. In ISO TC 171 SC 2, WG 5 owns ISO 19005 and WG 8 owns ISO 32000.

Term	Definition
Language Pack	A file or set of files which specify all string constants for a given language as well as additional localisation (such as date format)
Machine-readable Report	A structured report, independent of language and localization, generated for automated processing rather than human readability.
Metadata Fix	A simple fix of the PDF Metadata embedded in the PDF Document.
Metadata Fixer	A PREFORMA term for the component “which allows for simple fixes of the metadata embedded in the file, making them compliant with the standard specification”
Metadata Fixing Report	A Machine-readable Report generated by the Metadata Fixer containing details of Metadata Fixes carried out and any exceptions.
PDF Document	A Byte Sequence claiming conformance with ISO 32000-1:2008 (for PDF/A-2 and PDF/A-3) or to the Adobe specification of PDF 1.4 (for PDF/A-1).
PDF Document Extract	A programmatic model of a PDF Document created by parsing a PDF. The model encapsulates applicable PDF syntax, any PDF Metadata, plus details of PDF Features. The model can be serialised as a Machine-readable Report.
PDF Feature	Any property of the PDF Document or any of its structural elements such as pages, images, fonts, color spaces, annotations, attachments, etc.
PDF Features Report	A Machine-readable Report containing details about PDF Features including PDF Metadata and other available XMP metadata packages.
PDF Metadata	PDF document-level metadata stream containing the XMP package and the entries of the PDF Info dictionary.
PDF Parser	A software component that reads a PDF Document and constructs a PDF Document Extract.
PDF Validation TWG (TWG)	The Technical Working Group coordinated by the PDF Association and attended by industry members to discuss and decide matters pertaining to PDF Validation.
PDF/A Document	A Byte Sequence claiming conformance to a specific PDF/A Flavour.
PDF/A Flavor	PDF/A Part+Level. Possible PDF/A Flavors are: 1a, 1b, 2a, 2b, 2u, 3a, 3b, 3u.
PDF/A Identification	The part of the XMP metadata in a PDF Document that identifies the PDF/A Part (1, 2 or 3) and conformance Level (b, a, or u) to which the PDF Document claims to conform.
PDF/A Level	Level a, b, or u conformance as defined by the PDF/A Part.
PDF/A Part	Part 1: ISO 19005-1:2005 Part 2: ISO 19005-2:2011 Part 3: ISO 19005-3:2012
PDF/A Validation	The process of testing whether the PDF Features of a PDF Document conform to the requirements for a particular PDF/A Flavor. The PDF/A Validation process generates a Validation Report.

Term	Definition
PDF/A Validation Report	A Machine-readable Report containing the results (all errors and notifications) of PDF/A Validation.
Policy	Institutional acceptance criteria for long-term archiving and preservation of PDF Documents and their PDF Metadata, including requirements beyond those specified in the PDF/A standards.
Policy Check	The execution of a discrete Policy Test for a particular PDF Document. The Policy Checker carries out Policy Tests when enforcing a particular Policy Profile.
Policy Checker	A PREFORMA term for the component “which allows for adding acceptance criteria, always compliant with the standard specifications, that further differentiates the properties of the file. This might, for example, include limiting conformance to PDF/A-1b, or exclude files containing a certain type of image.”
Policy Profile	A file that expresses institutional Policy as a set of formal Policy Tests.
Policy Profile Registry	Enables the discovery and exchange of Policy Profiles between institutions.
Policy Report	A Machine-readable Report containing the results of Policy Checks performed as defined in a Policy Profile.
Policy Test	A Test Assertion that is evaluated by examining a PDF Features Report to ensure a PDF Document complies with institutional Policy.
PREFORMA Shell	A PREFORMA term for an interactive component: “the conformance checker should interface with other systems through a ‘shell’ which allows for interfacing multiple conformance checkers at the same time. This might in the future allow integrating the conformance checkers of different suppliers into one application.”
Report Template	A template file that defines the layout and format of a Human-readable or Machine-readable report. These are used by the Reporter to transform Machine-readable Reports into alternative formats.
Reporter	A PREFORMA term for the component that “interprets the output of the implementation checker and policy checker and allows for defining multiple human and machine readable output formats. This might include a well-documented JSON or XML file, a human readable report on which specifications are not fulfilled, or a fool-proof report which also indicates what should be done to fix the errors.”
SHA-1 Hash	A cryptographic hash function that creates a digital fingerprint for a byte sequence, referred to as ‘message’ in cryptographic documentation. A SHA-1 Hash value is 20 bytes, or 40 hexadecimal digits, long.
Test Assertion	A Test Assertion is a testable or measurable expression for evaluating the adherence of an implementation (or part of it) to a normative statement in a specification.
Validation Model	A formal definition of all PDF Document objects, their properties, and relationships between them expressed in a custom syntax.

Term	Definition
Validation Profile	A structured file describing the set of Validation Tests to be performed during PDF/A Validation for a particular PDF Flavour.
Validation Test	A Test Assertion that is evaluated by examining a PDF Features Report to ensure a PDF Document complies with a requirement expressed in a specific PDF/A Flavour.
veraPDF API	The veraPDF application programming interface defines the operations, inputs, outputs, and types that form the protocols for interacting programmatically with the veraPDF Library.
veraPDF Command Line Interface	The command line executable(s) providing access to the veraPDF Library API and functionality via a command line interface.
veraPDF Configuration	The detailed settings which configure an invocation of the veraPDF Conformance Checker. Configuration settings are logically divided into: ● **task config:** settings controlling the behaviour of a component, these are reusable across executions and installations; ● **installation config:** settings unique to a particular installation, such as home and temp directories; ● **execution config:** settings unique to a particular invocation, such as files or URLs to check, names of output report files.
veraPDF Framework	A software library that provides a lightweight framework based on open standards for use by Conformance Checker developers.
veraPDF Desktop Graphical User Interface (GUI-D)	An executable program that provides access to the veraPDF API and Library on a desktop computer or workstation.
veraPDF Library	The software library that provides the functionality and APIs for PDF/A Validation, Policy Checking, Metadata Fixing, and Reporting.
veraPDF REST API	RESTful web service API that provides HTTP access to the veraPDF Library functionality. This is a REST layer on top of the veraPDF API.
veraPDF Shell	Provides the user interfaces to manage and operate the Conformance Checker, handling issues such as workflow control and scheduling.
veraPDF Web Graphical User Interface (GUI-W)	Browser based HTML user interface that calls the veraPDF Library through the REST API, which in turn calls the veraPDF API.

Community Engagement

CE Introduction

In addition to developing software the veraPDF Consortium will undertake other activities supporting the terms of the PREFORMA tender, specifically:

- interact with industry experts and standards organisations for guidance and clarification in interpreting the relevant specifications;
- develop an open licensed corpus of test files that instantiates a reference interpretation of the PDF/A standards (see the PDF/A Test Corpus Report for an analysis of the coverage of existing corpora against the standard specifications);
- establish and foster an open source project and community of users and developers who will be the custodians of the software once the funded period is completed.

CE 1 Stakeholders

Refining the analysis in the veraPDF Tender Proposal section 1.1 I Stakeholders (p. 9), we identify key communities, stakeholders within those communities, their interest in the project, and the relationship with veraPDF Consortium members with respect to PREFORMA.

Table 1 describes the various community interests in the project and the aims of the veraPDF consortium which relate to each of those interests, and identifies the stakeholders and their stake in the project interests and aims.

Mechanisms for engaging veraPDF stakeholders are described in Annex A.		**Memory Institutions**		**Industry**			**3rd party communities**	**Research Organisations**	**Commercial Customers**
		Developers	Users	PDF vendors	Other software vendors	ISO	[tbd]	Researchers	Users
Awareness	Project visibility	x	x	x	x	x	x	x	x
	Updates on progress	x	x	x		x	x	x	
Recruitment	Identify collaborators	x	x	x		x	x	x	
Contribution	Functional requirements		x	x		x	x		x
	Technical requirements	x		x					x
	Corpora	x	x	x			x	x	
	Code	x		x			x		
	Documentation	x	x	x					
	Third-party extensions				x		x		

Mechanisms for engaging veraPDF stakeholders are described in Annex A.		**Memory Institutions**		**Industry**			**3rd party communities**	**Research Organisations**	**Commercial Customers**
		Developers	Users	PDF vendors	Other software vendors	ISO	[tbd]	Researchers	Users
Evaluation	Functional review		x	x		x		x	x
	Technical review	x		x		x		x	
	Software testing	x	x	x				x	x
Adoption	Implementation	x	x	x	x				x
	Support		x		x				x
	Sustainability		x		x				x

Table 1: *domains, stakeholders, interests and community objectives*

CE 2 Community development activities

CE 2.1 veraPDF ecosystem

PREFORMA Evaluation Criteria

D8.1. *(i) healthy ecosystem: the project establish a healthy ecosystem around an open source 'reference' implementation for specific file formats;*

Referring to the original veraPDF Tender Proposal, section 1.1 I *A sustainable ecosystem to ensure long-term sustainability* (pp. 8-10) this section describes the mechanisms of community interaction in more detail, highlighting how the proposed structures deliver on the objectives of the PREFORMA challenge. Specifically, we demonstrate how the complementary remits of the veraPDF consortium partners contribute to a healthy and long-lived ecosystem.

The Communications Plan, which describes the audiences and channels which will be addressed in more detail, can be found in Annex A: Communications Plan.

CE 2.2 Specific communities

In order to accomplish the objectives of the PREFORMA challenge, veraPDF engages and collaborates with a broad community of stakeholders.

CE 2.2.1 Industry and Standards

PREFORMA Evaluation Criteria

D8.1 *(vii) propose changes and additions: technology providers draft proposals for changes and additions to the standard specifications;*

D8.1 *(viii) participate in work-groups: technology providers participate in technical workgroups that maintain a standard specification;*

A key feature of the veraPDF value proposition lies in the claim of being **definitive**. We addressed the significance of definitive validation in the original veraPDF Tender Proposal, section 1.1 I *Methodology* (pp. 7-8). The definitive PDF/A validator is not only a faithful implementation of the standard, it is also the formal test corpora and associated software that possesses the quality of being *generally accepted* by the community for determining whether or not a PDF Document conforms to ISO 19005 requirements.

CE 2.2.1.1 Adoption factors

The PDF/A Competence Center published the Isartor Test Suite in 2008 which was used to support development of shipping products, with leading vendors such as Adobe Systems, callas software, intarsys, PDF Tools, LuraTech and SEAL Systems using it immediately for quality assurance.

As described in the original veraPDF Tender Proposal, section 1.1 I *Methodology* (pp. 7-8) the veraPDF Consortium leverages the PDF industry developer ecosystem as embodied in the PDF Association and its history of fostering understanding, adoption, and best-practices pertaining to PDF/A via the Isartor Test Suite and Technical Notes, its Technical Working Groups (TWG), and its formal 'Category A' liaison relationships to relevant ISO Working Groups (WGs).

In the context of veraPDF several factors will combine to drive rapid and general adoption of the veraPDF corpora and software across the various domains within the marketplace.

Key among these is the establishment by the PDF Association of the new PDF Validation Technical Working Group (TWG) which was created as part of the activities in Phase 1.

Given the scope of veraPDF - the corpora, software features, interpretation of the specifications, involved parties, purpose-built extensible design, development process, and promotional factors - we expect at least an equivalent rate of adoption, and certainly a broader reach, compared with the Isartor Test Suite.

CE 2.2.1.2 PDF Validation Technical Working Group (TWG)

To address PREFORMA Phase 1, the Board of the PDF Association elected to form a new TWG oriented towards the question of validation of PDF in general, including PDF/A in the context of veraPDF.

The role of the PDF Validation TWG is to:

- assemble interested parties to discuss strategy, policy and questions of interpretation;
- provide an international forum for establishing industry consensus on veraPDF test files, software messaging and message translation (localization or internationalization);
- provide a formal vehicle for recording decisions and driving veraPDF findings to developers;
- coordinate with the PDF and PDF/A TWGs and 3rd party organisations;
- request clarifications and propose changes directly to the responsible ISO WG.

As the formal venue within the PDF Association for validator scope and design, policy for interpretation of the PDF/A specifications and their instantiation as Validation Profiles, test file approval, and acceptance testing the PDF Validation TWG performs five unique roles which have the effect separately and together of promoting rapid acceptance and adoption of veraPDF industry-wide.

The TWG drives the degree to which veraPDF is regarded as definitive in several ways.

Feature	Relevant factor	Contribution to “definitive”
ISO WG relationship	The PDF Association is the category A liaison to the WGs in ISO TC 172 SC 2. Most members of the ISO 19005 WG are also represented in the TWG.	Substantial. The PDF Association has an established formal means of communication with the ISO WGs. Issues raised in the TWG may be brought directly to the attention of the relevant ISO WG.
Industry awareness	The TWG ensures that the industry technical community is aware of the validator.	Substantial. Awareness is vital to ensuring the broadest impact, fastest adoption rates and most consistent implementation.
Technical clarity and implementation diversity	TWG meetings, discussions, profiles, test files and results are available to all PDF Association members, ensuring broad consideration across a diverse set of implementers.	Substantial. The TWG provides a forum for questions and the means of reviewing, adjudicating, recording and publishing complex or substantive decisions regarding architecture, software functionality, Validation Profiles, test files, localization and more.

Feature	Relevant factor	Contribution to “definitive”
Industry leadership	TWG involvement by Adobe Systems, callas software, PDF Tools, and other ISO WG and PDF/A implementation leaders.	Vital. These companies are the industry leaders in creating and processing PDF/A files, including founding members of the PDF/A Competence Center.
Transparency	A clear development and test file selection process accepted by TWG members on a consensus basis.	Vital. True industry acceptance requires that contentious cases are resolved openly and by general acclamation including, if necessary, ISO WG review.
Focus	veraPDF is a purpose-built validator, not a PDF parser with validation features.	Modest. However, this approach allows the majority of effort to go towards the design objective rather than enablement.

Table 2: *veraPDF features driven by the PDF Validation TWG*

CE 2.2.1.3 Progress in Phase 1

48 PDF Association members, including a majority of the regular members of ISO TC 171 SC 2 WG 5 (PDF/A) joined the PDF Validation TWG mailing list to participate in Phase 1. Developers in many time zones who do not attend the meetings (they are regularly scheduled at 1700 CET) watch recordings of the meetings, which include any slides or other documents presented, and audio of the discussion.

The developer of the PDF/A validator licensed by Adobe Systems for use in PDF/A conversion and validation in Adobe Acrobat is a vocal member of the TWG, as is Adobe Systems’ “PDF Architect” (the company’s lead developer on standards conformance matters and the ISO 19005 Project Leader).

The PDF Validation TWG operates with the objective of finding consensus. To help facilitate consensus-based outcomes, the TWG established in December 2014 a Validation Advisory Board (VAB) made up of expert developers. If the VAB fails to resolve a dispute the TWG may refer items to the PDF Association Board or the respective ISO WG for resolution.

The PDF Validation TWG convenor and responsible PDF Association staff member for coordinating the industry response to the PREFORMA challenge is a regular member since 2007 of the ISO 19005 committee, and has served as ISO Project Leader for ISO 32000 since 2010.

PDF Validation TWG meeting agendas to-date have included:

- TWG structure and process, introduction of TWG Chair and Validation Advisory Board;
- Overview of the Functional Specification draft and test methods;
- Top level architecture;
- Validation profile and test suites;
- The scope of PDF/A validation with respect to external specifications;
- Presentation of the veraPDF validation profile model;
- Validation of embedded formats:
    - The updated list of all embedded formats relevant for PDF/A validation;
    - Validation of embedded ICC profiles;
    - Validation of embedded fonts;
- Comparison with DVA and Levigo PDF formal presentation format;

- The validation algorithm;
- Using Xtext for the formal syntax;
- Validating "number" in PDF/A-1 inconsistencies in the requirements for TrueType built-in encoding;
- Strategy for validating PDF/A Level A (Tagged PDF):
    - distinguishing between machine- and human-verifiable conditions;
    - parts of PDF 1.4 / ISO 32000-1:2008 specifications that shall be validated.

CE 2.2.2 Other domains / communities / standards

As discussed in the original veraPDF Tender Proposal, section 1.1 II *Potential of the Proposed Idea / Solution / Technology to Address Future and/or Wider Challenges in the Area* (pp. 10-14), a purpose-built open source validator for a format with the visibility and importance of PDF provides substantial potential and opportunity for community members and others to leverage the work of the veraPDF Consortium and apply it to related technologies. A key enabler of this potential is the inherent extensibility of the software architecture. As designed (see FS 3 Conformance Checker extensions) the veraPDF Conformance Checker will provide an attractive framework for other validators as it facilitates their use in the PDF context, a vast arena.

The veraPDF Validation Model (see TS 2 Validation Model) does not preclude 3rd party engines for 3rd party purposes. In addition, the model is not linked to any platform or specific development technology. A font developer, for example, might build their own font program validator using the veraPDF architecture to better understand the encoding of subsets in PDF Documents. In our view, this fact increases prospects that the veraPDF model is sufficiently generic to itself become a *de facto* standard for “definitive” validation of Byte Streams in a multi-vendor environment.

Such an architecture may even be ultimately necessary to the project of a definitive validator. Beyond PDF 1.4 and ISO 32000, PDF/A specifies requirements in 3rd party standards. Our maximally generic approach is intended to facilitate outreach, communication, and cooperation with the respective stakeholders in these technologies.

CE 2.2.2.1 Specific extensions

Conformance with PDF/A requires conformance with applicable components of various 3rd party specifications (see FS 1 PDF/A Validation in context). veraPDF will reach out to stakeholders in these 3rd party communities to encourage participation in the veraPDF community and encourage the consideration of the veraPDF generic, purpose-built validator to inspire their own validator development efforts, ideally aligning them directly with veraPDF.

External specification	Owner / community	Planned activity
Colour profiles	International Colour Consortium (ICC)	Personal contact via PDF Association members.
TrueType	Adobe Systems, font developers	Personal contact via PDF Association members.
ISO/IEC 14496 (OpenType)	Microsoft, font developers	Personal contact via Microsoft contacts.

External specification	Owner / community	Planned activity
ISO/IEC 14496 (Open Font Format)	Various, including Microsoft	Personal contact via PDF Association members.
ISO/IEC JTC 1/SC 29 (JPEG 2000)	The JP2 imaging community	Personal contact via PDF Association members.
	Jpylyzer (hosted by OPF)	Personal contact via OPF.
ISO 16684 (XMP)	XMP metadata community	Personal contact via PDF Association members.

Table 3: *3rd party communities and planned activities*

CE 2.2.2.2 Impact on extensibility

The veraPDF approach enables a wide variety of supporting and parallel collaborations.

Feature	Relevant factor	Contribution to "extensible"
Flexible software architecture	An open, generic design facilitates a wide variety of implementation scenarios.	Substantial. Provides a framework for greenfields development of any given component, increasing flexibility and lowering technical barriers to entry.
PDF parser agnostic	Although the proposed reference implementation of veraPDF will leverage a specific low-level parser the architecture will be library-agnostic. (see FS 3.1.1.1 PDF Parsers).	Substantial. Allows implementers to integrate veraPDF with their preferred creation or processing libraries .This strategy future-proofs the software, encouraging continued development in diverse implementations.
Generic plugin architecture	The veraPDF model encourages plug-ins for parsing not only PDF/A-related third-party data structures (see FS 1.2.1 PDF/A requirements beyond PDF syntax), but also for other features in ISO 32000, other ISO standards for PDF such as PDF/E or PRC, images, and for embedded content such as rich media or attachments (see FS 3.1.1.2 Embedded Resource Parsers).	Although limited under PREFORMA funding to addressing PDF/A, veraPDF will encourage development of other components to validate objects that may appear in PDF/A Documents but are out of scope for PREFORMA. The plugin model facilitates broad-based efforts to develop and promote file format validators. For example, Jpylyzer may be adapted as a veraPDF plugin analyzing JPEG 2000 images in PDF Documents.

Table 4: *veraPDF features pertaining to extensibility*

CE 2.2.2.3 Progress in Phase 1

In Phase 1 work has been limited to identifying external dependencies and the community organisations which control the relevant standards and specifications. Until Phase 2 is funded approaching these communities would not result in meaningful collaboration, however this will be a priority for the start of Phase 2 once we can announce that work on veraPDF is in progress.

CE 2.2.3 Memory institutions

PREFORMA Evaluation Criteria

> D8.1 (ix) *facilitate OAIS Monitor Designated Communities: the network of common interest enables implementation of the OAIS Monitor Designated Communities function for Preservation Planning, interacting with Archive Consumers and Producers to track changes in their service requirements and available product technologies;*
>
> D8.1 (x) *facilitate OAIS Develop Preservation Strategies and Standards: the network of common interest enables implementation of the OAIS Develop Preservation Strategies and Standards function for preservation planning, developing and recommending strategies and standards, and for assessing risks, to enable the Archive to make informed trade-offs as it establishes standards, sets policies, and manages its system infrastructure;*
>
> D8.1 (xi) *facilitate OAIS Establishing Standards and Policies: the network of common interest enables implementation of the OAIS Establishing Standards and Policies function by the Administration of the Archive system and maintain them.*

FS 2.5 veraPDF Shell describes how the veraPDF Conformance Checker software acts as a component within an OAIS-archive, specifically enabling processes associated with Ingest. FS 2.3 veraPDF Policy Checker describes the mechanism for creating and using Policy Profiles which in turn leads to the creation of a Policy Profile Registry, described below. In addition, the veraPDF consortium and community further enable additional functions of the OAIS-archive, also described below.

CE 2.2.3.1 Registry of Policy Profiles

The Conformance Checker functionally separates the application of a Policy Profile from other aspects of the operation (such as technical environment and execution variables) which logically separates Policy Profiles from other aspects of the local environment and makes it possible to reuse a Policy Profile created by another institution. The mechanism and data formats for expressing Policy Profiles are described in the Technical Specification (see TS 5 Policy profile).

In order to facilitate the sharing of Policy Profiles between institutions, veraPDF will create a Policy Profile Registry. The Registry will provide a means of discovering, obtaining, and publishing Policy Profiles, to include:

- a web interface for searching and browsing existing Policy Profiles based on the description of the institutional policy provided by the author (for example by PDF/A Part or feature, or external format such as image or font);
- downloadable Policy Profiles in the defined data format, created as a result of community requirements gathering and representing common institutional policies;
- user guides (documentation) describing how to use and update Policy Profiles;
- a mechanism for uploading and sharing newly created Policy Profiles.

During Phase 2, memory institutions will be invited to contribute policy requirements and veraPDF will provide support in expressing these in the formal language of Policy Profiles. We will encourage the sharing of these exemplar Policy Profiles for testing and reuse by other institutions, both providing quality assurance of the policies and allowing common policy requirements to be identified across the community. In turn, this will impact other OAIS-archive functions as described below.

CE 2.2.3.2 Impact on OAIS-archive functions

The veraPDF model and approach to community development enable several OAIS functions pertaining to non-technical aspects of an OAIS-archive.

OAIS function	veraPDF feature	Impact on memory institutions
Monitor Designated Communities	Industry adoption	Improves understanding of the market for PDF software, including commonly used creating and editing suites, providing detailed information about available technologies for Producers and Consumers.
	Definitive validation	Enables the definition of explicit service levels for deposit and access taking into account format validity.
Develop Preservation Strategies and Standards	Corpora which comprehensively instantiate the requirements of PDF/A	Enables the comprehensive analysis of format functionalities on the basis of authoritative information and test files which can be used for testing and evaluating preservation strategies, leading to an understanding of risk in the content of preservation planning.
Establish Standards and Policies	PDF Validation TWG	Enables communication of policy checking requirements to the relevant ISO WG for consideration (e.g. highlighting ambiguities in the specifications).
	Policy Profiles and the Policy Profile Registry	Enables sharing of best practice and lowers barriers to implementing institutional policies by sharing common requirements and test files.

Table 5: *features pertaining to OAIS functions*

CE 2.2.3.3 Progress in Phase 1

The Open Preservation Foundation ran a webinar for members presenting the Functional and Technical Specifications for review. Several institutions provided detailed feedback on both the mechanisms for expressing and enforcing policy and on specific policy requirements of their institutions. These are given as examples in FS 2.3 Policy Checker.

CE 3 Contribution guidelines

This section refers to and extends the open source practices as described in the original veraPDF Tender Proposal, section 1.1 IV *Cohesion with open source development values and objectives* (pp. 15-18). Specifically, it provides details about how the community will be managed, and how code, files, and documentation can be contributed and the quality criteria that will be applied by the open source project leader (OPF Technical Lead). Contribution guidelines will be published on the veraPDF website (and other appropriate locations) to provide support for the community.

CE 3.1 Functional and Technical Specifications

The Functional and Technical Specifications will be published openly at the start of Phase 2. At this point they will have been subjected to community review by the PDF Association Validation TWG and Open Preservation Foundation members as described in CE 2.2 Specific communities.

Revisions to these documents will be made during Phase 2 on the basis of community engagement including face-to-face events and mailing lists, for example new uses cases or technical requirements, or updates to existing uses cases or technical requirements. There will be a formal change management process which will require the veraPDF Consortium to publish new versions of the documents and update the development roadmap where required.

The redesign stage of Phase 2 is anticipated to produce a revised major edition (e.g. Functional or Technical Specification version 2) while incremental editions updating or refining single use cases or requirements (e.g. versions 1.1 or 2.1) may be published at any time.

During Phase 3 management of these documents and the development roadmap will be turned over to the open source community. All historical versions of all documents will be available through the veraPDF website at all times.

CE 3.2 Corpora

PREFORMA Evaluation Criteria

> D8.1 *(ii) demonstration files: technology providers contribute demonstration files with good and bad samples of the corresponding reference implementation;*

Corpora will be produced for PDF/A Validation (one for each PDF/A Flavour), Policy Checking, and Metadata Fixing (see TS 6.2 Test files). All corpora will be built and managed using community contributions and will be subject to a formal submission and review process. In each case the review will be the responsibility of the veraPDF partner with expertise and authority, for Validation Corpora this will be the PDF Association Validation TWG as described in CE 2.2.1.2 PDF Validation Technical Working Group (TWG).

Formal contribution agreements will be drafted and will be required from any submitter who wishes to contribute to the corpora. The agreements will require the submitters to license their contributions under the required open licences (see the original veraPDF Tender Proposal section V *Test corpora* and our response to the Negotiation Report 2. *Adherence to licence requirements for all digital assets developed during the PREFORMA project*). Contributions will not be accepted without the formal agreement which will have to be signed by a designated person within each institution with authority to sign on behalf of the institution (for example company CEO or library director).

All corpora will be managed using Git for revision control and the Git repositories will be publically available on the veraPDF GitHub organisation page. Test files will not be added directly to the test corpora by any

individual. Instead, borrowing from software development best-practice, anyone wishing to extend a corpus will first clone the test corpus repository.

Working in a local branch a contributor can add test files to the corpus. The repository README file for each corpus will express corporus-specific acceptance criteria for submission, e.g file naming, repository structure, technical guidelines, or accompanying documentation. Submissions must also observe these general principles for all corpora:

- the contributor creates an issue describing the test case on the corpora GitHub issue tracker, this should be as fine-grained / atomic as possible;
- each test file should demonstrate a pass or fail case for the atomic issue;
- no more than five test files should be added to the repository in a single commit;
- each commit has a descriptive comment that states what the committed files are.

Each submission should be made as a GitHub pull request to the veraPDF corpus repository. The pull request should connect the issue addressed with the test files committed using GitHub flavoured markdown. The pull request will be examined by the body responsible for the particular corpus repository submissions and reviewed objectively using the submission criteria. If accepted, the pull request will be merged into the veraPDF corpus repository. If the pull request can't be merged the reviewer will inform the contributor of the reasons and suggest appropriate changes before resubmission of the request.

CE 3.2.1 Validation Corpora

The PDF Association Validation TWG will review and approve each candidate for inclusion in the Validation Corpora. As described above, this will ensure their authority as an objective frame of reference by involving domain experts and members of the standards committees in approving the test files as the authoritative realization of the PDF/A specifications.

CE 3.2.2 Policy Checking Corpus

The Open Preservation Foundation will review and approve each candidate in the Policy Checking corpus and provide support to early adopters during Phase 2 in expressing Policy Requirements as formal Policy Profiles.

To be accepted for testing in the prototyping phase, policy requirements must consist of:

- a textual statement of the policy, supplied by the institution;
- an owner, usually an individual from the institution who has authored the policy;
- formal rule(s) (see TS 5.2 Using Schematron for Policy Checks);
- test files that express pass and fail cases for inclusion in the corpus.

Policy Profiles and the associated Policy Checking corpus will be available through the Policy Profile Registry (see CE 2.2.3.1 Registry of Policy Profiles).

CE 3.2.3 Progress in Phase 1

The PDF Association Validation TWG assisted in identifying gaps in existing corpora as described in Annex C: PDF/A Test Corpora Report.

CE 3.3 Code

CE 3.3.1 Code acceptance

Coding standards will be automatically enforced through the projects build system. The Maven build has PMD and Checkstyle plugins that detect and report code and coding style issues beyond just compilation errors. All pull requests will be automatically checked firstly using these tools and secondly for test coverage as defined in TS 6.4.1 Unit testing. If the contribution does not meet these criteria then a reviewer - at first the open source project leader until other formal roles are assigned within the community - will contact the contributor, via the pull request chain on GitHub, identifying improvements which are necessary for the submission to be accepted.

Formal contribution agreements will be drafted and will be required from any submitter who wishes to contribute to the software. The agreements will require the submitters to license their contributions under the required open source licences (see Annex E: License Compatibility). Contributions will not be accepted without the formal agreement which will have to be signed by a designated person within each institution with authority to sign on behalf of the institution (for example company CEO or library director).

CE 3.4 Messaging

Validation results are delivered to veraPDF users in messages. These messages may appear in user interfaces and Machine-readable or Human-readable Reports.

To most openly and effectively align industry interests with veraPDF, and thus maximize acceptance of the software, the PDF Association Validation TWG will oversee and approve the Implementation Checker messages and translations thereof (see TS 7 Internationalization).

Software instructions help files, and other operational content will be open to broader community input.

CE 3.5 Documentation

PREFORMA Evaluation Criteria

> D8.1 *(iii) documentation of the source code: technology providers contribute comprehensive documentation of the source code, which allows for automated generation of the internal API of the application;*
>
> D8.1 *(iv) documentation of the software: technology providers contribute comprehensive documentation of the conformance checker for developers, such as quick start guide, cook- books and other tutorials;*
>
> D8.1 *(v) online technical support: technology providers ensure online availability at the development platform for technical support to other developers deploying the conformance checker;*

The original veraPDF Tender Proposal section 1.1 IV *Documentation* (p. 17) describes the different types of documentation, their audience, and responsible author in detail. The only addition to this is the production of a Frequently Asked Questions to be maintained through the project web presence responding to common queries raised by the community. Where appropriate, these inquiries will feed into other aspects of the development, for example as feature requests updating the Functional or Technical Specifications, or as other contributions such code, corpora, documentation, or testing.

Since veraPDF is developed and documented openly in front of a commercially-interested community, it is anticipated that not only will the fundamental code and documentation quality be closely monitored, but the

precise messaging of the software, especially with respect to validation, will be the subject of intense scrutiny in the PDF software industry. Documentation drafts will be available with early releases of the prototype for testing and will be subject to revision and update based on community feedback.

veraPDF will be developed with all documentation, UI elements and software interactions in English. The initial implementation will demonstrate support for a limited number of European languages to demonstrate the localization mechanism (see TS 7 Internationalization).

Functional Specification

FS Introduction

Section 1 describes PDF/A Validation in context, taking a detailed look at the ISO specifications and defining the scope of the Implementation Checking functionality. Section 2 describes the functionality of the Conformance Checker components and how they are co-ordinated by the Shell to satisfy the PREFORMA challenge. Section 3 describes the extensibility of the Conformance Checker. Section 4 describes the user interfaces, providing detailed examples of command line invocation of the Shell.

FS 1 PDF/A Validation in context

This section describes the relevance to validation of *shall*, *should*, and *may* statements within ISO standards and details how PDF/A (ISO 19005) relates to PDF (ISO 32000) and other 3rd party standards.

The following attributes are key principles of the veraPDF Conformance Checker:

- Definitive validation against the requirements specified in all parts and conformance levels of PDF/A (ISO 19005-1, 19005-2 and ISO 19005-3) including the 2007 and 2011 Technical Corrigenda to ISO 19005-1.
- Extensibility to cover non-native data-structures and features of PDF (ISO 32000) not addressed in PDF/A (see FS 3.1 Parsing PDF Documents and Embedded Resources);
- Industry acceptance - because the software is designed, built, tested, and assessed, in front of leading PDF software developers (see CE 2.1 The veraPDF ecosystem).

FS 1.1 'Shall', 'should', and 'may' statements

veraPDF defines the relevance to validation of statements within standard specifications as follows:

- The software will address "*shall*" and "*shall not*" statements in the Implementation Checker, as compliance with these statements is required for normative PDF/A Validation (see Annex C: PDF/A Test Corpora Report);
- The software will address "*should*", and "*may*" statements in the Policy Checker, as these statements do not affect normative PDF/A Validation (see Annex C.3 PDF/A "Should" and "May" Clauses);
- In addition to file format requirements, PDF/A includes requirements for "conforming reader" software to ensure consistent rendering of text and graphics. As visual rendering is out of scope in a file format validator, veraPDF does not address requirements for a conforming reader.

FS 1.2 PDF/A, PDF, and associated standards and specifications

The specification for PDF/A is a set of restrictions and requirements applied to the "base" PDF standards (PDF 1.4 for PDF/A-1 and ISO 32000 for PDF/A-2 and PDF/A-3) plus a specific set of 3rd party standards (see Table 1).

PDF files may include many other types of data structures. Apart from those defined in ISO 32000 itself, that specification includes 80 third-party documents as normative references.

Where necessary to the goals of PDF/A, ISO 19005 identifies the specific clauses within either PDF 1.4 or ISO 32000, or specific third-party documents, to which conformance requirements apply. Involvement from these 3rd party communities is anticipated (see CE 2.2.2 Other domains / communities / standards).

FS 1.2.1 PDF/A requirements beyond PDF syntax

As noted above, PDF/A directly references 3rd party data structures defined elsewhere, including images, fonts, ICC profiles and more. These are enumerated fully in the table below. Addressing validity criteria in

these 3rd party standards is required for a definitive PDF/A conformance checker.

Although the software design will be extensible to any feature of PDF, and to any type of object that may be utilized or contained in PDF files, the PREFORMA-funded veraPDF Implementation Checker will check only those requirements made explicit in the text of ISO 19005. Clauses not explicitly required in ISO 19005 are considered out of scope.

Some examples:

- **JPEG2000:** veraPDF will refer to ISO 32000-1 for validation of JPEG2000 compressed objects because PDF/A-2 clause 6.2.8.3 states: “JPEG2000 compression shall be used as specified in ISO 32000-1:2008.” However, veraPDF will not validate JPEG2000, only the manner in which such objects are encoded as per ISO 32000;
- **Fonts:** veraPDF will determine whether font widths are encoded consistently (as is required in PDF/A-2 clause 6.2.11.5), however since deeper font validation is not explicitly required in PDF/A it will not check other font data such as glyph outlines.
- **ICC Profiles:** veraPDF will check the header section of the embedded ICC profiles to make sure the profile version and class conform to the PDF/A specifications (for example, PDF/A-2 clauses 6.2.3 and 6.2.4.2), but it will not perform the complete validation of the embedded ICC stream against relevant ICC specifications (ICC.1:1998-09, ICC.1:2001-12, ICC.1:2003-09 or ISO 15076-1);
- **Tagged PDF:** the literal requirements in PDF 1.4 and ISO 32000-1:2008 for Tagged PDF are very limited, and PDF/A does not expand significantly on these requirements for its conformance level a. Accordingly, veraPDF will perform only those limited checks on Tagged PDF files as are required by PDF/A and relevant sections of PDF 1.4 and ISO 32000-1:2008 specifications (see Annex C.2 Tagged PDF Test Suite for the full list of Tagged PDF clauses and the corresponding test cases).

The veraPDF strategy for 3rd party specifications referenced by PDF/A is defined in the following table.

Feature	PDF/A	3rd Party Spec	Strategy
ICC color profiles	**PDF/A-1:** All ICCBased colour spaces shall be embedded as ICC profile streams as described in PDF Reference 4.5. **PDF/A-2:** The profile that forms the stream of an ICCBased colour space shall conform to ICC.1:1998-09, ICC.1:2001-12, ICC.1:2003-09 or ISO 15076-1. **PDF/A-3:** As PDF/A-2	ICC.1:1998-09; ICC.1:2001-12; ICC.1:2003-09; ISO 15076-1	Read and validate profile version number, Device Class signature, Color Space of Data in the ICC profile header, but do not validate other data. Provide third-party plug-in mechanism for custom ICC profile validation.

Feature	PDF/A	3rd Party Spec	Strategy
Image compression	**PDF/A-1:** The LZWDecode filter shall not be permitted. **PDF/A-2:** All standard stream filters listed in ISO 32000-1:2008, 7.4, Table 6 may be used, with the exception of LZWDecode **PDF/A-3:** As PDF/A-2	ITU Recommendations T.4 and T.6; JBIG2 Specification; ISO/IEC 10918 (JPEG), Adobe Technical Note #5116, Supporting the DCT Filters in PostScript Level 2	Provide plug-in mechanism for third party tools to validate the compressed images, but do not validate them internally.
JPEG2000	**PDF/A-1:** not permitted **PDF/A-2:** JPEG2000 compression shall be used as specified in ISO 32000-1:2008. Only the JPX baseline set of features, as restricted or extended by ISO 32000-1:2008 and this subclause, shall be used. **PDF/A-3:** As PDF/A-2	ISO/IEC 15444-2	Check all Color Specification boxes ('colr'), Image Header Box ('ihdr') and check the absence of Bits Per Component box ('bpcc'). Validate the embedded ICC profile as described above, if the ColorSpace key in the PDF Image dictionary is absent. Provide third-party plug-in mechanism for custom JPEG2000 validation, but do not perform any deeper validation internally.
Font embedding	**PDF/A-1:** The font programs for all fonts used within a conforming file shall be embedded within that file, as defined in PDF Reference 5.8. **PDF/A-2:** All fonts and font programs used in a conforming file, regardless of rendering mode usage, shall conform to the provisions in ISO 32000-1:2008, 9.6 and 9.7, as well as to the font specifications referenced by these provisions. **PDF/A-3:** As PDF/A-2	Adobe Type 1 Font Format; TrueType Reference Manual; Adobe Technical Note #5176; ISO/IEC 14496-22:2009	Parse widths and validate the presence of the glyphs used in PDF page content; validate the presence of the required tables in TrueType/ OpenType fonts, check that the font is legally embedded, but do not validate any further details of font program Provide third-party plug-in mechanism for font validation.

Feature	PDF/A	3rd Party Spec	Strategy
CJK encodings, Unicode character maps	**PDF/A-1:** All CMaps used within a conforming file, except Identity-H and Identity-V, shall be embedded in that file as described in PDF Reference 5.6.4. **PDF/A-2:** All CMaps used within a PDF/A-2 file, except those listed in ISO 32000-1:2008, 9.7.5.2, Table 118, shall be embedded in that file as described in ISO 32000-1:2008, 9.7.5. **PDF/A-3:** As PDF/A-2	Adobe Technical Note #5014 Adobe CMap and CIDFont Files Specification	Validate the complete CMap stream and its consistency with other PDF data.
Digital signatures	**PDF/A-1:** Not specified **PDF/A-2:** As permitted by ISO 32000-1:2008, 12.8.1, a PDF/A-2 conforming file may contain document, certifying or user rights signatures. Such signatures shall be specified in the PDF through the use of signature fields in accordance with ISO 32000-1:2008, 12.7.4.5. **PDF/A-3:** As PDF/A-2	PDFA TN0006 - Digital Signatures in PDF/A-1; RFC 3280, Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile	Validate the presence of keys and values prescribed by ISO 32000-1:2008 but do not perform the cryptographic validation of the signature. Provide the third-party plug-in mechanism to validated the embedded PKCS#1 and PKCS#7 certificates.
Metadata	**PDF/A-1:** All metadata streams present in the PDF shall conform to the XMP Specification. All content of all XMP packets shall be well-formed, as defined by Extensible Markup Language (XML) 1.0 (Third Edition), 2.1, and the RDF/XML Syntax Specification (Revised). **PDF/A-2:** As PDF/A-1 **PDF/A-3:** As PDF/A-1	XML 1.0 (W3C Recommendation 04 Feb 2004); RDF/XML Syntax Specification (W3C Recommendation 10 Feb 2004); XMP Specification	Parse all embedded XMP packages at all levels. Validate that XMP metadata is well-formed according to W3C schema for RDF/XML. Validate all requirements on permitted XMP schemas as well as the consistency with Info dictionary. Provide third-party plug-in mechanism for metadata format validation.

Feature	PDF/A	3rd Party Spec	Strategy
Attached files	**PDF/A-1:** Not permitted **PDF/A-2:** Only PDF/A-1 and PDF/A-2 attachments are permitted **PDF/A-3**: allows for embedding of files of any type, but imposes certain requirements for embedded files that go beyond what ISO 32000-1 requires.	Any (depends on the attached file)	**PDF/A-1:** None. **PDF/A-2:** validate attachments (PDF/A attachments only). **PDF/A-3:** Validate PDF/A attachments, but do not validate them internally. Provide third-party plug-in mechanism for external format validation.

Table 1: *validation required by explicit references beyond the text of the PDF/A specifications*

Deeper analysis of the definitive validation for embedded ICC profiles and Fonts is provided in Annex G ICC Profile Checks for PDF/A Validation and Annex H Embedded Font Checks for PDF/A Validation. They serve mainly as an overview of the complexity for the definitive validation of PDF/A including all embedded files and are subject to review and collaboration with the experts in the corresponding areas (see CE 2.2.2 Other domains / communities / standards).

To encourage the alignment of 3rd party development efforts with veraPDF, the plugin mechanism facilitates collaboration with 3rd party technology communities (see FS 3 Conformance Checker extensions).

FS 1.2.2 What PDF/A is not

PDF/A concentrates on key matters of interest to memory institutions, introducing restrictions on usage of the larger PDF specifications (PDF 1.4 for PDF/A-1 and ISO 32000 for PDF/A-2 and PDF/A-3). PDF/A does not account for every possible reason why a PDF Document may be unusable in whole or in part and by itself is not a panacea for any possible problem with archivable electronic documents.

The table below gives examples of factors that may impact Document reliability but are out of scope for PDF/A validation purposes. These examples demonstrate the types of problem; it is not an exhaustive listing of all possible non-PDF/A issues relating to Document reliability.

Out of scope for PDF/A validation	Possible consequence
Corrupt images (e.g. JPEG2000 or other).	Image content may not be legible.
Corrupt or poorly subsetted font programs.	Text content may not be legible.
Invalid data relevant to high-end printing (printer's marks, device colorant data, trapping support and other features specific to high-end print implementations).	Inconsistent results in high-end printing.
Content outside the PDF page crop box.	Content may be present (and thus, searchable) but not displayed on the page.
Invalid encoding of semantics in the document's logical structure.	The document may not be reliably repurposed using PDF logical structure mechanisms.

Table 2: *Examples of potential reliability concerns not addressed by PDF/A*

FS 2 Conformance Checker components

This section describes the modular components that make up the veraPDF Conformance Checker.

FS 2.1 veraPDF Implementation Checker

The Implementation Checker parses and analyzes PDF Documents. It outputs two types of report: a PDF Features Report describing the PDF Document and its PDF Metadata and a Validation Report describing conformance to PDF/A Flavours.

FS 2.1.1 Use cases

FS 2.1.1.1 Generate a PDF Features Report

A user requests a report describing the details of features found in a PDF Document (including its metadata, Document information such as number of pages, and information about embedded files such as fonts, images, or color spaces). The user doesn't want to establish whether the PDF Document conforms to a PDF/A Flavour but wants a Machine-readable Report that can be stored in a repository system for later use by the Policy Checker (for example when enforcing a new institutional policy or analysing the content of a repository). This report is used by the Policy Checker and its formatting is handled by the Reporter.

Input	Byte Sequence believed to be a PDF Document
Output	PDF Features Report
Options	The user can pass the PDF Features Report for Policy Checking or serialise the Report to a Machine-readable or Human-readable format via the Reporter.
Extensions	The Reporter enables transformations of the PDF Features Report (see FS 2.4 veraPDF Reporter)
Exceptions	If the Byte Sequence cannot be identified as a PDF Document or is too malformed to be parsed successfully then the Implementation Checker will report this as an error.

FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour

A user requests a report detailing the conformance of a PDF Document with a PDF/A Flavour and listing all errors. The formatting of the Validation Report is handled by the Reporter.

Input	Byte Sequence believed to be a PDF Document
Output	Validation Report
Options	The user must pass a parameter specifying the PDF/A Flavour. The Implementation Checker loads the Validation Profile relating to the PDF/A Flavour. The user can pass a parameter instructing the Implementation Checker to stop processing after a set number of errors have occurred.
Extensions	The Reporter enables transformations of the Validation Report (see FS 2.4 veraPDF Reporter)

Exceptions	If the Byte Sequence cannot be identified as a PDF Document or is too malformed to be parsed successfully then the Implementation Checker will report this as an error.

FS 2.1.2 Functional description

The veraPDF Implementation Checker provides the following functionality:

- parsing a PDF Document to generate a PDF Features Report;
- parsing the PDF Document and Validation Profile to check conformance to a PDF/A Flavour and generate a Validation Report.

The Implementation Checker relies on a PDF library for PDF parsing (see FS 3.1.1.1 PDF Parsers). The original veraPDF Tender Proposal described two approaches: development of a greenfield PDF Parser built from scratch and an option to use PDFBox. The important architectural point is that our design is 'parser-agnostic': the PDF Parser can be changed without affecting the functionality of the Implementation Checker.

Implementation Checking relies on parsing the PDF Document and running Validation Tests defined in the Validation Profiles for each PDF/A Flavour. The general model for format validation is described in TS 2 Validation Model and the Validation Profile format is defined in TS 3 Validation Profile format. Validation Profiles will be supplied for all PDF/A Flavours covered in the PREFORMA Challenge and subjected to the review of the PDF Association Validation TWG (see CE 2.2.1 Industry and Standards).

The Implementation Checker generates a PDF Features Report and a Validation Report. The complete report format for both types of Report is defined in TS 4 Machine-readable Report format.

The PDF Features Report includes all information about the PDF Document. In summary, this includes:

- PDF Metadata:
  - the information dictionary;
  - all available XMP metadata packages;
- Low-level PDF information, including:
  - document-level information: Document ID, date/time stamps; number of indirect objects, trailer keys; output intent; conformance claims; trapping; number of pages; colorants;
  - page level information: bounding boxes, resources, annotations;
  - resource information: the type of resource and its properties;
  - annotation information;
  - form fields information;
  - details of filters, encryption, digital signatures.

The Validation Report gives the results of PDF/A Validation. In summary, this includes:

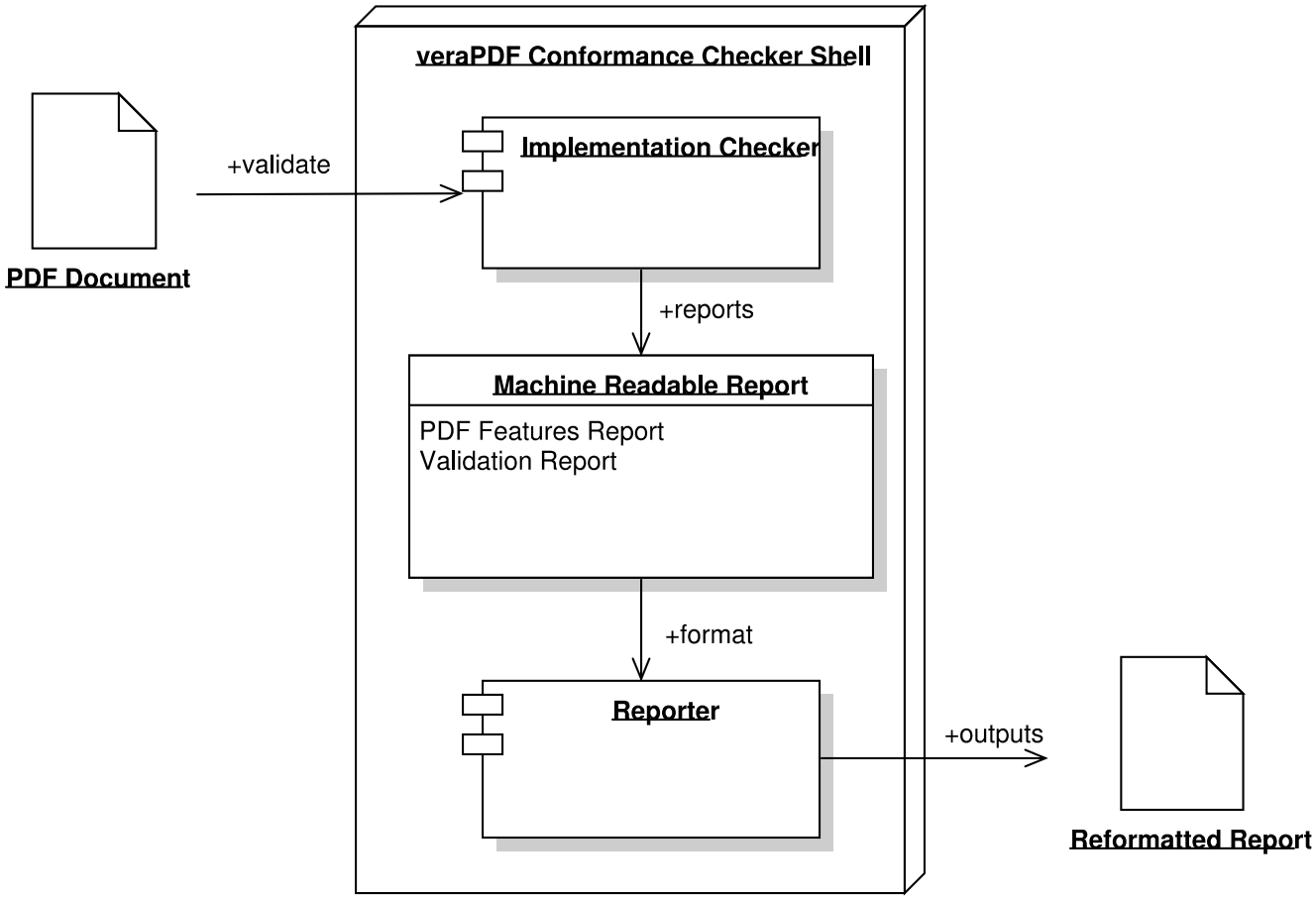
- pass/fail on conformance against the specified PDF/A Flavour;
- detailed results from tests of all normative statements in the PDF/A specifications ("shall", "should" and "may" statements) according to the chosen PDF/A Flavour.

The Validation Checks are carried out at the following levels:

- File I/O requirements for the low-level PDF syntax as such as binary header format, spacing and end of line symbol requirements;
- PDF syntax rules for required/forbidden/recommended/permitted keys and their respective value types are performed after the PDF document is opened;

- Higher level tests (metadata, output intent, color space compliance, etc) requiring complex logic not limited to a single key/value pair in the PDF dictionary;
- Graphics content tests are checked by parsing the content streams of the following objects:
    - page content streams for all pages present in the page tree;
    - annotation appearance streams;
    - form field appearance streams (widget annotations);
    - content streams for the Form XObject, Tiling Pattern, and Type3 Font resource types;
- Checking for the glyphs present in an embedded font file, this first entails collecting all glyphs used in the document. The check is an example of a post-action, which must be deferred until all checks they depend upon are complete;
- Checks for Embedded Resources governed by the external specifications (images, fonts, colour profiles) are performed as described in FS 1.2.1 PDF/A requirements beyond PDF syntax and FS 3.1.1.2 Embedded Resource Parsers).

FS 2.1.3 Functional architecture



The user passes a PDF Document to the Conformance Checker. The Implementation Checker generates a PDF Features Report. The Implementation Checker processes the PDF Document against the Validation Profile corresponding to the requested PDF/A Flavour and generates a Validation Report. Both reports are passed to the Reporter for reformatting.

FS 2.2 veraPDF Metadata Fixer

The Metadata Fixer makes well-defined, discrete fixes to PDF Metadata within PDF Documents so that it complies with a PDF/A Flavour. The Metadata Fixer produces a Repaired PDF Document that is a fixed version of the original and a Metadata Fixing Report which describes the fixes attempted and their success or failure.

FS 2.2.1 Use cases

FS 2.2.1.1 Remove invalid PDF/A Metadata and produce a new PDF Document

A user requests that a PDF Document is checked for validity against the PDF/A Flavour claimed in the PDF Metadata and requests that the PDF Metadata is fixed based on the results of the Validation. A report on the Metadata Fixing is requested and its formatting is handled by the Reporter

Depending on the results of the Validation, the possible outcomes are:

- If the PDF Document fails Validation but claims PDF/A Flavour conformance in the PDF Metadata then a Repaired PDF Document is created without the conformance claim in the PDF Metadata;
- If the file passes validation but the PDF Document does not claim PDF/A Flavour conformance in the PDF Metadata then a Repaired PDF Document is created with the conformance claim in the PDF Metadata;
- In other cases (a PDF Document which correctly claims or does not claim PDF/A Flavour conformance) then nothing is done (a Repaired PDF Document is not produced but a Metadata Fixing Report is still produced).

Input	PDF Document Validation Report
Output	Repaired PDF Document Metadata Fixing Report
Options	To overwrite the input stream, where possible.
Extensions	The Reporter enables transformations of the Metadata Fixing Report (see FS 2.4 veraPDF Reporter)
Exceptions	If the PDF Metadata is malformed (but embedded correctly) the Metadata Fixer may be unable to repair the PDF Document. In this case, the Metadata Fixer reports an error.

FS 2.2.1.2 Fix PDF Metadata and produce a new PDF Document

A user requests that a PDF Document is checked for validity against a specified PDF/A Flavour and requests that the PDF Metadata is fixed based on the results of the validation. A Validation Profile identifies possible fixes for each Validation Test and these are passed to the Metadata Fixer. A report on the Metadata Fixing is requested and its formatting is handled by the Reporter.

Input	PDF Document Validation Report
Output	Repaired PDF Document Metadata Fixing Report
Options	To overwrite the input stream, where possible.
Extensions	The Reporter enables transformations of the Metadata Fixing Report (see FS 2.4 veraPDF Reporter)
Exceptions	If the PDF Metadata is malformed (but embedded correctly) the Metadata Fixer may be unable to repair the PDF Document. In this case, the Metadata Fixer reports an error.

FS 2.2.2 Functional description

The veraPDF Metadata Fixer performs a predefined set of actions to correct problems affecting the PDF Metadata of an otherwise valid PDF/A Document, or removing the PDF/A Metadata in the case of a PDF Document that does not conform to PDF/A. The Metadata Fixer always creates a Repaired PDF Document and leaves the decision to overwrite the original in the hands of the user. The Metadata Fixer generates a Metadata Fixing Report that describes the fixes carried out and details of the Repaired PDF Document or describes any problems encountered if the file could not be fixed.

Fixes attempted by the Metadata Fixer will include the following, which have been subject to review by the PDF Validation TWG as described in CE 2.2.1.2 PDF Validation Technical Working Group (TWG).

FS 2.2.1.1 Remove invalid PDF/A Metadata and produce a new PDF Document

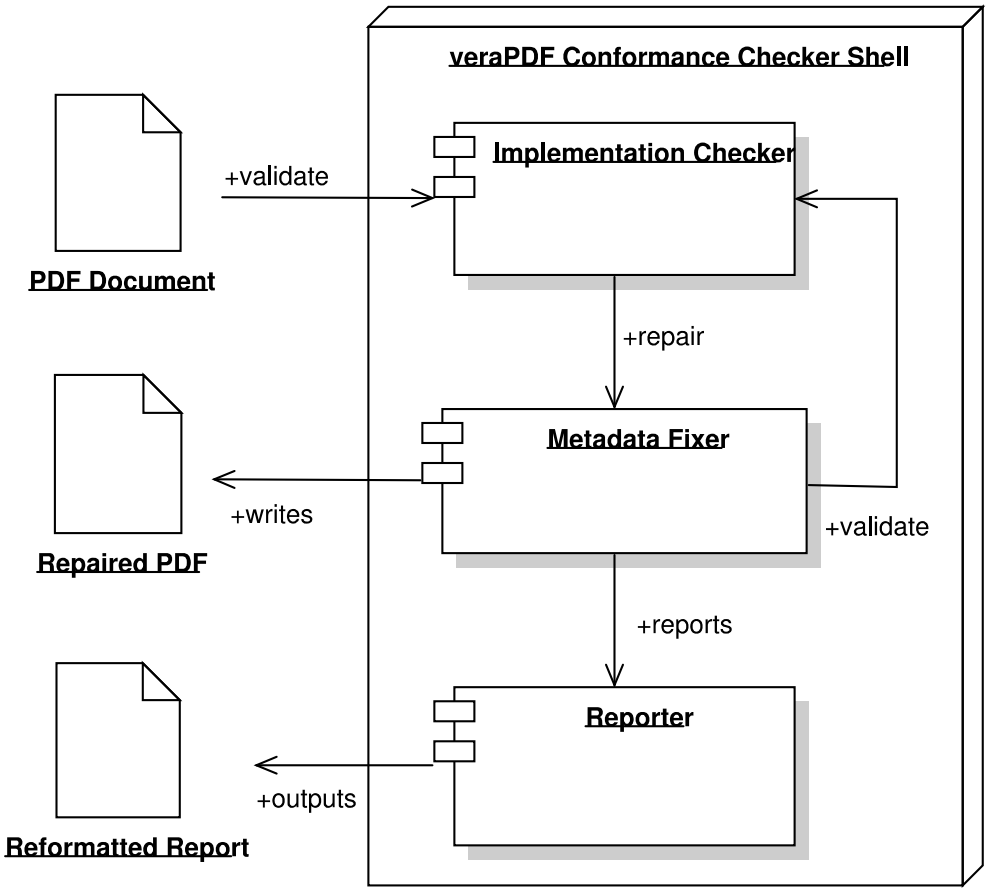
- adding PDF/A Identification to an otherwise valid PDF/A Document;
- removing PDF/A Identification from a PDF document that fails the validation.

FS 2.2.1.2 Fix PDF Metadata and produce a new PDF Document

- synchronizing Info dictionary with document XMP Metadata (only for PDF/A-1);
- adding default XMP package or predefined schemas if they are missing in the PDF Metadata.

The Metadata Fixer will also provide an interface for third-party modification of XMP packages.

FS 2.2.3 Functional architecture



A PDF Document is passed to the Conformance Checker for Validation and Metadata Fixing. The Implementation Checker processes the PDF Document producing a PDF Features Report and Validation Report. The Validation Profile identifies possible fixes which are passed to the Metadata Fixer to be attempted. The Metadata Fixer produces a Repaird PDF Document and a Metadata Fixing Report detailing the success/failure of each fix. The new PDF Document is passed back to the Implementation Checker which produces a new PDF Features Report and Validation Report. These new reports, and the Metadata Fixing Report, are added to the Machine-readable Report and passed to the Reporter.

Note that for the sake of simplicity the Machine-readable Reports have been left out of this diagram.

FS 2.3 veraPDF Policy Checker

The Policy Checker parses and analyzes a PDF Features Report and generates a Policy Report stating whether the PDF Document complies with institutional policy as expressed in a Policy Profile.

Some examples of institutional policy statements could include:

- require all “should” and “may” clauses in the PDF/A specifications to be enforced;
- accept PDF/A-2 files but reject embedded images in a specific format (e.g. JPEG 2000 which is permitted by PDF/A-2);
- disallow documents containing particular fonts (e.g. ComicSans) or particular types of fonts (e.g. TrueType) even if they are embedded as specified by PDF/A;
- accept PDF/A-3 files but restrict the permissible formats of attachments (e.g. accept only CSV attachments for a particular collection);
- disallow the use of particular types of embedded XMP metadata, OR insist that a particular type of XMP metadata is present, in line with submission criteria.

Institutional policy requirements will be gathered, prototyped as Policy Profiles, and made available to the community as described in CE 2.2.3 Memory institutions.

FS 2.3.1 Use cases

FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements

A user requests a report detailing whether a PDF Document conforms to institutional policy. They must supply their policy requirements as a formal Policy Profile. They may supply the PDF Document itself to the Conformance Checker to produce a PDF Features Report as described in FS 2.1.1.1 Generate a PDF Features Report or they may supply an existing PDF Features Report retrieved from storage, for example a repository. The formatting of the Policy Report is handled by the Reporter.

Input	PDF Features Report for the PDF Document Policy Profile (expressing institutional policy as formal rules)
Output	Policy Report
Options	The user can pass a parameter instructing the Policy Checker to stop processing after a set number of errors have occurred
Extensions	The user may integrate Embedded Resource Parsers making an Embedded Resource Report available to the Policy Checker (see FS 3.1.1.2 Embedded Resource Parsers) The Reporter enables transformations of the Policy Report (see FS 2.4 veraPDF Reporter)
Exceptions	If a PDF Features Report cannot be generated as described in FS 2.1.1.1 Generate a PDF Features Report then a Policy Report cannot be generated.

FS 2.3.1.2 Author a new Policy Profile

A user wishes to express their institutional policy as a Policy Profile so that the Policy Checker can check the conformance of a PDF Document as described in FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements. They may create the Policy Profile at any time, regardless of whether PDF Documents are to be processed by the Conformance Checker. They may change their policy requirements and produce a new Policy Profile which can be used to process PDF Documents via the Implementation Checker or existing PDF Features Reports which have been generated and stored as described in FS 2.1.1.1 Generate a PDF Features Report.

Note that authoring a Policy Profile does not depend on the Conformance Checker *per se*, but on an external editor. The user may require technical skills to author the Policy Profile according to the required format which is specified in TS 5 Policy Profile.

Input	Institutional policy (expressed as free text)
Output	Policy Profile (expressing institutional policy as formal rules)

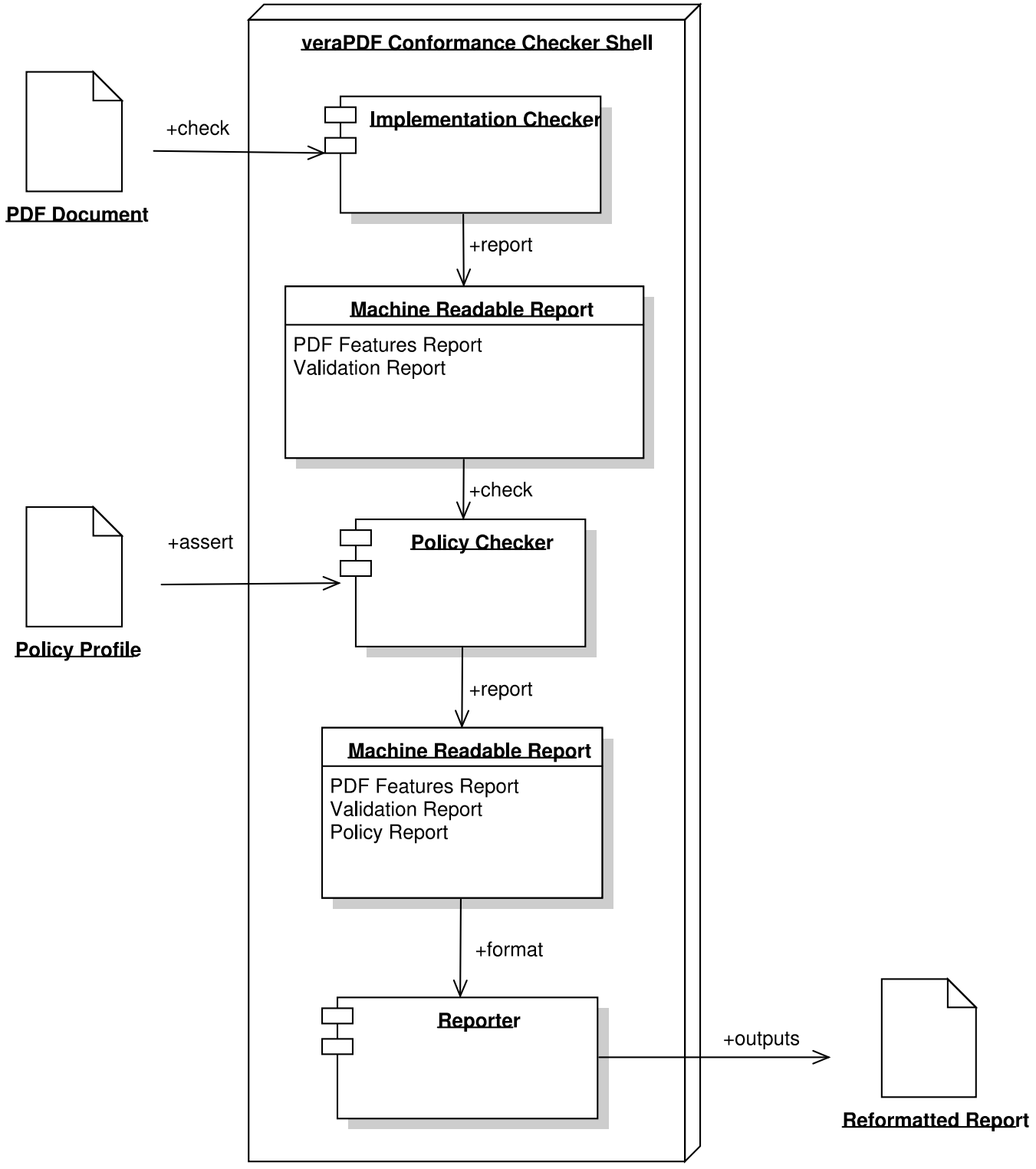
Options	Creating a Policy Profile can be done using a simple text editor or using a format-specific editor as described in TS 5 Policy Profile.
Extensions	Sharing Policy Profiles between institutions is enabled by the Policy Profile Registry, see CE 2.2.3.1 Registry of Policy Profiles. Test files demonstrating pass/fail of the Policy Checks can be submitted as described in CE 3.2.2 Policy Checking Corpus.
Exceptions	A malformed Policy Profile will cause the Policy Checker to report an error.

FS 2.3.2 Functional description

The Policy Checker processes a PDF Features Report applying the rules expressed in a Policy Profile and generating a Policy Report. The Policy Report contains information about the conformance of the PDF Document against institutional policy requirements. The format of the report is described in TS 4 Machine-readable Report format.

The Policy Checker is independent of the Implementation Checker to the extent that it can operate on the PDF Features Report without invoking the PDF/A Validation functionality of the Implementation Checker.

FS 2.3.3 Functional architecture



FS 2.4 veraPDF Reporter

The Reporter transforms the Machine-readable Reports generated by the Implementation Checker, Policy Checker, and Metadata Fixer into other forms. It is supplied with standard Report Templates and users can define their own. Reports can be Human-readable (including HTML and PDF) or Machine-readable (including XML and other formats supported by external systems such as workflows or repositories).

FS 2.4.1 Use cases

FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy, Metadata Fixing)

A user wishes to obtain a Machine-readable Report containing one or more of:

- PDF Features Report as described in FS 2.1.1.1 Generate a PDF Features Report;
- Validation Report as described in FS 2.1.1.2 Generate a Validation Report;
- Policy Report as described in FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- Metadata Fixing Report as described in FS 2.2.1.1 Remove invalid PDF Metadata and produce a new PDF Document and FS 2.2.1.2 Fix PDF Metadata and produce a new PDF Document.

The user chooses a predefined Report Template or supplies a custom Report Template (see FS 2.4.1.4 Author a new Report Template). They may configure a verbosity level to control the amount of information in the Report.

Input	Machine-readable Report (PDF Features, Validation, Policy, Metadata Fixing) Report Template
Output	Machine-readable Report in the specified format
Options	The user may specify a verbosity level to control the amount of information contained in the Machine-readable Report.
Extensions	Report Templates enable the generation of reports in custom formats. This can be used to output to JSON, PREMIS (XML), SWORD, or other formats understood by external systems such as workflow managers or repository systems. Internationalisation enables the translation of of Machine-readable Reports into languages other than English, see TS 7 Internationalization.
Exceptions	A malformed Report Template will cause the Reporter to report an error.

FS 2.4.1.2 Obtain a Human-readable Report (PDF Features, Validation, Policy, Metadata Fixing)

A user wishes to obtain a Human-readable report containing one or more of:

- PDF Features Report as described in FS 2.1.1.1 Generate a PDF Features Report;
- Validation Report as described in FS 2.1.1.2 Generate a Validation Report;
- Policy Report as described in FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- Metadata Fixing Report as described in FS 2.2.1.1 Remove invalid PDF Metadata and produce a new PDF Document and FS 2.2.1.2 Fix PDF Metadata and produce a new PDF Document.

The user chooses a predefined Report Template or supplies a custom Report Template (see FS 2.4.1.4 Author a new Report Template). They may configure a verbosity level to control the amount of information in the Report.

Input	Machine-readable Report (PDF Features, Validation, Policy, Metadata Fixing) Report Template
Output	Human-readable Report
Options	The user may specify a verbosity level to control the amount of information contained in the Human-readable Report.
Extensions	Report Templates enable the generation of reports in custom formats. This can be used to output to HTML, PDF, or other user-specified formats. Internationalisation enables the translation of of Machine-readable Reports into languages other than English, see TS 7 Internationalization. Report Templates supplied by veraPDF will be consistent with accessibility best practices as described in TS 8.2 Accessibility.
Exceptions	A malformed Report Template will cause the Reporter to report an error.

FS 2.4.1.3 Obtain Machine-readable or Human-readable Reports for a batch of PDF Documents

A user wishes to obtain a Machine-readable Report or Human-readable report as described in FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy) or FS 2.4.1.2 Obtain a Human-readable Report (PDF Features, Validation, Policy) but for a batch of PDF Documents. They may request a summary of multiple Machine-readable or Human-readable Reports, for example summarising PDF Features, Validation Checks, or Policy Checks across the batch.

Input	Machine-readable Reports (PDF Features, Validation, Policy, Metadata Fixing) Report Template
Output	Machine-readable Report and/or Human-readable Report
Options	The user may specify a verbosity level to control the amount of information contained in the Machine-readable or Human-readable Report.
Extensions	Report Templates enable the generation of reports in custom formats. This can be used to outputs to HTML, PDF, or other user-specified formats. Internationalisation enables the translation of of Machine-readable Reports into languages other than English, see TS 7 Internationalization. Report Templates supplied by veraPDF will be consistent with accessibility best practices as described in TS 8.2 Accessibility.
Exceptions	A malformed Report Template will cause the Reporter to report an error.

FS 2.4.1.4 Author a new Report Template

A user wishes to define a new format for Machine-readable or Human-readable Reports so that they can be produced as specified in FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy), FS 2.4.1.2 Obtain a Human-readable Report (PDF Features, Validation, Policy), or FS 2.4.1.3 Obtain Machine-readable or Human-readable Reports for a batch of PDF Documents.

They may create the Report Template at any time, regardless of whether any Reports are to be processed by the Reporter. They may change their formatting requirements and produce a new Report Template which can be used to process reports generated via the Implementation Checker, Policy Checker, or Metadata Fixer or to process existing PDF Features Reports, Validation Reports, Policy Reports, or Metadata Fixing Reports which have been generated and stored previously.

Note that authoring a Report Template does not depend on the Conformance Checker *per se*, but on an external editor. The user may technical skills to author the Report Template according to the required format which is specified in TS 8 Report Template format.

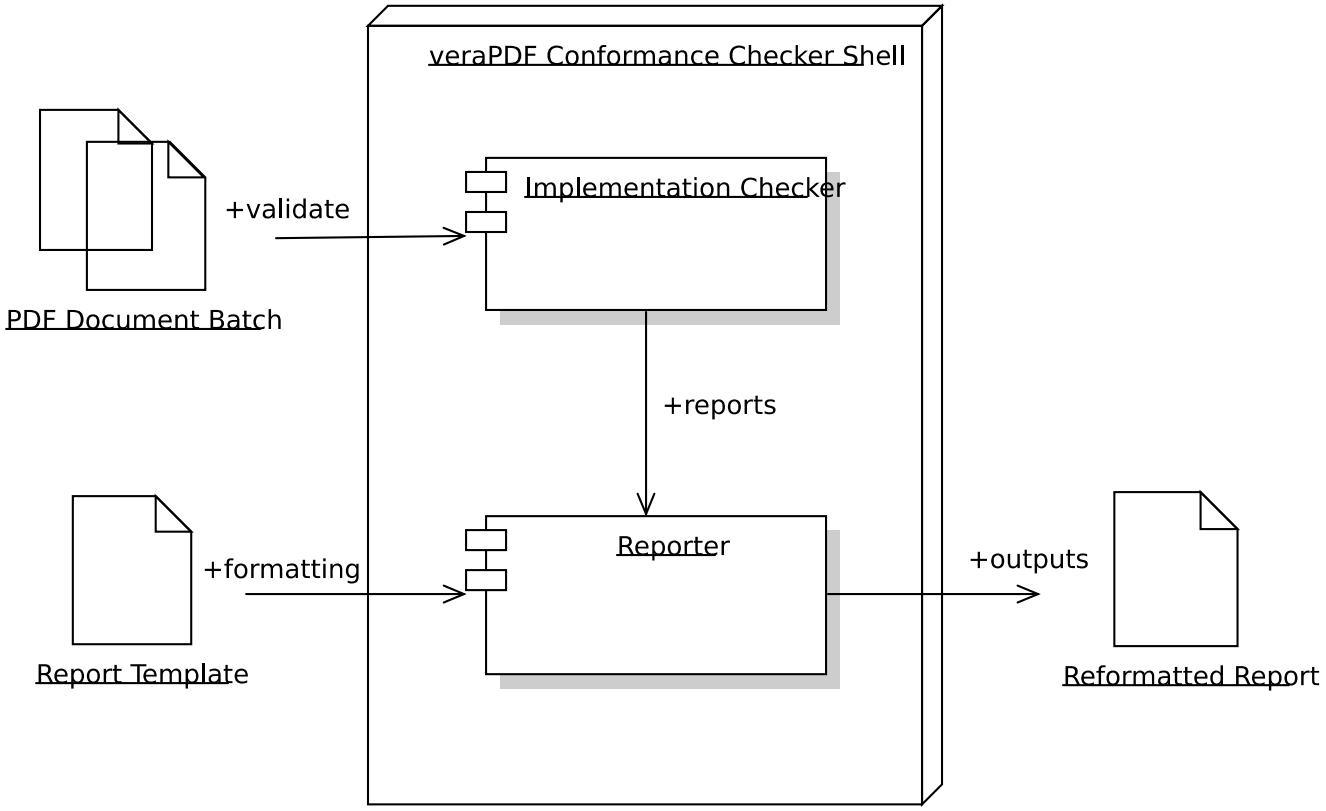
Input	Formatting requirements (such as the schema for a Machine-readable format or design preferences expressed as free text)
Output	Report Template (expressing formatting requirements as formal rules)
Options	Creating a Report Template can be done using a simple text editor or using a format-specific editor as described in TS 8 Report Template format.
Extensions	A Report Template can operate on any information contained in Machine-readable Reports. This may include information produced by 3rd-party parsers as described in TS 9 Integration with third-party tools.
Exceptions	A malformed Report Template will cause the Reporter to report an error.

FS 2.4.2 Functional description

The Reporter parses and transforms Validation Reports from the Implementation Checker, Policy Reports from the Policy Checker, and Metadata Fixing Reports from the Metadata Fixer. It outputs Machine-Readable Reports and Human-readable Reports according to Report Templates.

The Machine-Readable Report format is defined in TS 4 Machine-readable Report format and Report Templates are defined in TS 8 Report Template format.

FS 2.4.3 Functional architecture



One of more PDF Documents are passed to the Conformance Checker. Machine-readable Reports are generated by the Implementation Checker, Policy Checker, or Metadata Fixer (for simplicity, the diagram shows only the Implementation Checker but the reporter also handles input from the Policy Checker and Metadata Fixer). A Report Template is applied to the Machine-readable Reports and a reformatted Machine-readable Report or Human-readable Report is generated. In batch mode, the reformatted reports can include summaries across Machine-readable Reports generated from multiple PDF Documents (see FS 2.5.1.5 Automated, periodical, or batch Conformance Checking).

FS 2.5 veraPDF Shell

The Shell manages the other components and their interaction, providing coordinated sequences of actions. Users interact with the Shell through the Command Line Interface, Desktop Graphical User Interface, or Web Graphical User Interface as described in FS 3 Interfaces.

FS 2.5.1 User stories

User stories for the Shell describe higher-level scenarios which require the coordination of Conformance Checker components by the Shell. These user stories combine use cases of the other components into sequences of actions.

FS 2.5.1.1 Conformance Checking at Digitization

A digitization studio operator is producing PDF Documents from photographed content and wants to ensure that the files they produce satisfy established acceptance criteria. The operator may be based at an internal digitization studio at a memory institution or an external digitization supplier who has been contracted on the basis of a tender which defines the acceptance criteria. They request a summary report to be submitted along with their project documentation or invoice.

For example, the acceptance criteria may specify:

- conformance to a PDF/A Flavour;
- embedded images in a certain format;
- a summary report in a human-readable format.

This user story combines the following use cases:

- FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour;
- FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- FS 2.4.1.2 Obtain a Human-readable Report (PDF Features, Validation, Policy).

Input	PDF Document(s) Policy Profile (optional) Report Template (optional)
Output	Human-readable Report (containing a summary of the PDF Document(s))
Options	The user may choose to run the Conformance Checker over a directory of files using the batch mode (see FS 2.5.1.7 Batch or periodical Conformance Checking)
Extensions	The digitization studio operator may choose to integrate the Conformance Checker into a digitization workflow manager (such as Goobi). They may do this using the Library API or the REST API. They may supply a Report Template for transforming Machine-readable Reports into formats understood by the workflow manager.
Exceptions	[see exceptions for the Conformance Checker components]

FS 2.5.1.2 Conformance Checking at Creation Time

A content producer wants to ensure that PDF Documents generated by office suites or managed by an Electronic Document and Records Management System (EDRMS) conform to established acceptance criteria. They may want to be alerted when a file is produced or uploaded to an EDRMS that does not meet the criteria.

For example, the acceptance criteria may specify:

- conformance to a PDF/A Flavour;
- the use of only certain fonts;
- a notification should alert a nominated person on file upload.

This user story combines the following use cases:

- FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour;
- FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy).

Input	PDF Document(s) Policy Profile (optional) Report Template (optional)

Output	Machine-readable Report (containing Validation Report(s) and/or Policy Report(s))
Options	The content producer may run the Conformance Checker in batch or periodical mode to process multiple documents at defined intervals. The content producer may invoke the Metadata Fixer to attempt fixes to PDF Documents. They may choose to overwrite the original or keep both versions.
Extensions	The content producer may supply a Report Template to transform the Validation Report and/or Policy Report into a format understood by an EDRMS. The content creator may integrate the Conformance Checker into an EDRMS to manage automated checking on file uploads and handle notifications to a content manager. The content creator may require that office suites procured for use within their organisation use veraPDF to check PDF Documents as they are created.
Exceptions	[see exceptions for the Conformance Checker components]

FS 2.5.1.3 Pre-submission Conformance Checking by Content Producers

A content producer wants to check a submission for conformance to established acceptance criteria. They want to include the results of the Conformance Checker in the Submission Information Package (SIP). They may want to attempt fixes to the PDF Documents and include fixed files in the submission (either duplicating or replacing the existing files).

For example, the acceptance criteria may specify:

- conformance to a PDF/A Flavour;
- the use of only certain fonts;
- machine-readable reports generated in a format understood by the target submission system.

This user story combines the following use cases:

- FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour;
- FS 2.2.1.2 Fix PDF Metadata and produce a new PDF Document;
- FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- FS 2.4.1.3 Obtain Machine-readable or Human-readable Reports for a batch of PDF Documents.

Input	PDF Document(s) Policy Profile (optional) Report Template (optional)
Output	Machine-readable report (containing Validation Report(s) and/or Policy Report(s) and/or Metadata Fixing Report(s)) in a format that can be included in the SIP. Fixed PDF Document(s)
Options	The content producer may invoke the Metadata Fixer to attempt fixes to PDF Documents. They may choose to overwrite the original or keep both versions.

Extensions	The user may supply a Report Template to transform the Machine-readable Report into a specific format for use in the SIP (for example PREMIS, ISAD(G), BagIt). The content producer may integrate the Conformance Checker into another system which manages the SIP creation and/or transfer to the archive.
Exceptions	[see exceptions for the Conformance Checker components]

FS 2.5.1.4 Conformance Checking at transfer

A user at a memory institution wants to check a received Submission Information Package (SIP) for conformance to established acceptance criteria. The SIP may contain one or more PDF Documents. The user must unpack the SIP (for example if it is encoded in whole or in part using TAR or METS) and pass the PDF Documents to the Conformance Checker.

For example, the acceptance criteria may specify:

- conformance to a PDF/A Flavour;
- forbidding images in certain formats;
- machine-readable reports generated in a format understood by the institution's repository system.

This user story combines the following use cases:

- FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour;
- FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- FS 2.4.1.3 Obtain Machine-readable or Human-readable Reports for a batch of PDF Documents.

Input	PDF Document(s) Policy Profile (optional) Report Template (optional)
Output	Machine-readable report or Human-readable Report (containing Validation Report(s) and/or Policy Report(s) and/or Metadata Fixing Report(s))
Options	The user may invoke the Metadata Fixer to attempt fixes to PDF Documents. They may choose to overwrite the original or keep both versions.
Extensions	The user may choose to automate the Conformance Checker based on notifications from a transfer manager (for example an application watching for changes to a file system location or a success message from an FTP client).
Exceptions	[see exceptions for the Conformance Checker components]

FS 2.5.1.5 Archival Information Update at Ingest

A user at a memory institution wants to generate detailed information about PDF Documents within a Submission Information Package and include this information with the Archival Information Package.

This user story combines the following use cases:

- FS 2.1.1.1 Generate a PDF Features Report;
- FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy).

Input	PDF Document(s)
Output	Machine-readable Reports (containing any of Validation, Policy, or Metadata Fixing)
Options	The user may invoke the Implementation Checker or Policy Checker to produce a Validation or Policy Report and include these in the AIP. The user may invoke the Metadata Fixer to attempt fixes to PDF Documents. They may choose to overwrite the original or keep both versions.
Extensions	The user may choose to integrate the Conformance Checker into an existing repository system (for example Archivematica or DSpace). The user may choose to supply a Report Template to transform the Machine-readable Reports into formats compatible with their repository system.
Exceptions	[see exceptions for the Conformance Checker components]

FS 2.5.1.6 Conformance Checking at migration

A user at a memory institution wishes to migrate an arbitrary set of files to PDF/A according to institutional format policy. The user controls the migration process using separate software (for example an office suite or PDF editing application) and requests that the migrated PDF Document is checked for conformance to a PDF/A Flavour and/or Policy Profile. If both documents are PDF Documents (i.e. a PDF to PDF/A migration, or a migration between PDF/A Flavours) the user may request two PDF Features Reports so they they can compare significant properties to determine whether the file has been altered in unacceptable ways (for example losing pages or images). In this case, the user must compare the two PDF Features Reports using separate software to look for changes.

This user story combines the following use cases:

- FS 2.1.1.1 Generate a PDF Features Report;
- FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour;
- FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- FS 2.4.1.3 Obtain Machine-readable or Human-readable Reports for a batch of PDF Documents;
- FS 2.4.1.4 Author a new Report Template.

Input	PDF Document(s) Report Template (optional)
Output	Machine-readable Reports (containing any of Validation, Policy, or Metadata Fixing)
Options	The user may invoke the Metadata Fixer to attempt fixes to PDF Documents. They may choose to overwrite the original or keep both versions.

Extensions	The user may choose to integrate the Conformance Checker into an existing repository system (for example Archivematica or DSpace). The user may choose to supply a Report Template to transform the Machine-readable Reports extracting significant properties for comparison.
Exceptions	[see exceptions for the Conformance Checker components]

FS 2.5.1.7 Batch or periodical Conformance Checking

For any other use case, the user may want to process a batch of PDF Documents and/or run the Conformance Checker at a set time (e.g. 28 February 2015) or at predefined intervals (e.g. weekly). The PDF Documents may be accessed through a file system, web server (URL), EDRMS or repository system, or API. The user may want to integrate the Conformance Checker into an external system which coordinates the execution ('pushing' PDF Documents into the Conformance Checker) or they may want the Conformance Checker to coordinate the execution ('pulling' PDF Documents from an external source by file system location, URL, or API using identifiers).

Input	PDF Document(s) Policy Profile (optional) Report Template (optional)
Output	Machine-readable Report (containing any or all of PDF Features Report, Validation Report, Policy Report, or Metadata Fixing Report) Human-readable Report (containing any or all of PDF Features Report, Validation Report, Policy Report, or Metadata Fixing Report)
Options	The user may control the verbosity level in aggregated reports to produce a summary of information across all PDF Documents in a batch.
Extensions	The user may choose to integrate the Conformance Checker into a legacy system (see FS 3.2 Integrations with other software).
Exceptions	[see exceptions for the Conformance Checker components]

FS 3 Conformance Checker extensions

FS 3.1 Parsing PDF Documents and Embedded Resources

FS 3.1.1 Use cases

FS 3.1.1.1 PDF Parsers

As described in FS 2.1 veraPDF Implementation Checker the veraPDF Conformance Checker is reliant on a PDF Parser to implement the PDF Validation Model, produce the PDF Document Extract, and generate a PDF Features Report (which is itself required to generate other reports including Policy Reports).

Given the PREFORMA licensing requirements we will develop a greenfields PDF Parser from scratch, see Annex E.4.1 Implementation Checker for a full discussion of our approach.

In addition, to demonstrate the modularity of our design we will collaborate with the PDFBox community to provide a version of the Conformance Checker which swaps out our greenfield PDF Parser with the PDF Parser provided by PDFBox. This will demonstrate that other PDF Parsers (including existing commercial or proprietary solutions) can be used without affecting the functionality of the Implementation Checker and other veraPDF components.

Note that this does not introduce a dependency on PDFBox - the Conformance Checker will still be licensed in accordance with the PREFORMA requirements. Neither will this involve development effort by veraPDF, the use of PDFBox will be handled under the provision of community support for adopters and not involve development effort on the part of veraPDF Consortium members, both demonstrating the technical design and the community engagement approach as described in CE 2.2.2 Other domains / communities / standards.

Input	Byte Sequence believed to be a PDF Document
Output	PDF Document Extract Embedded Resource(s)
Options	None
Extensions	The Implementation Checker produces a PDF Features Report from the PDF Document Extract as described in FS 2.1.1.1 Generate a PDF Features Report. The user may integrate an Embedded Resource Parser to process the Embedded Resource and produce an Embedded Resource Report as described in FS 3.1.1.2 Embedded Resource Parsers.
Exceptions	If the Byte Sequence cannot be identified as a PDF Document or is too malformed to be parsed successfully then the Conformance Checker will report this as an error.

FS 3.1.1.2 Embedded Resource Parsers

A user wants to carry out detailed analysis of Embedded Resources within a PDF Document and obtain more detailed information than that provided in a PDF Features Report. A developer integrates an Embedded Resource Parser which can handle the appropriate formats. The user may create and provide a Policy Profile that defines institutional policy for features of Embedded Resources.

Institutional policy relating to Embedded Resources include:

- images must be 300 dpi grayscale or a minimum of 400 pixels wide;
- images, fonts, or colour profiles must be valid according to the relevant standard;
- attachments must be in a certain format and contain certain metadata.

Input	Embedded Resource extracted from the PDF Document
Output	Embedded Resource Report
Options	The user may create a Policy Profile which specifies Policy Checks to be applied to the Embedded Resource Report by the Policy Checker.
Extensions	Embedded Resource Parsers can be integrated for any conceivable type of Embedded Resource as long as that they are compatible with the interaction interface defined in TS 9 Integration with third-party tools.
Exceptions	An Embedded Resource Parser could fail to parse an Embedded Resource or return the information expected by the user. In this case the Conformance Checker will report an error and not provide an Embedded Resource Report.

FS 3.1.2 Functional description

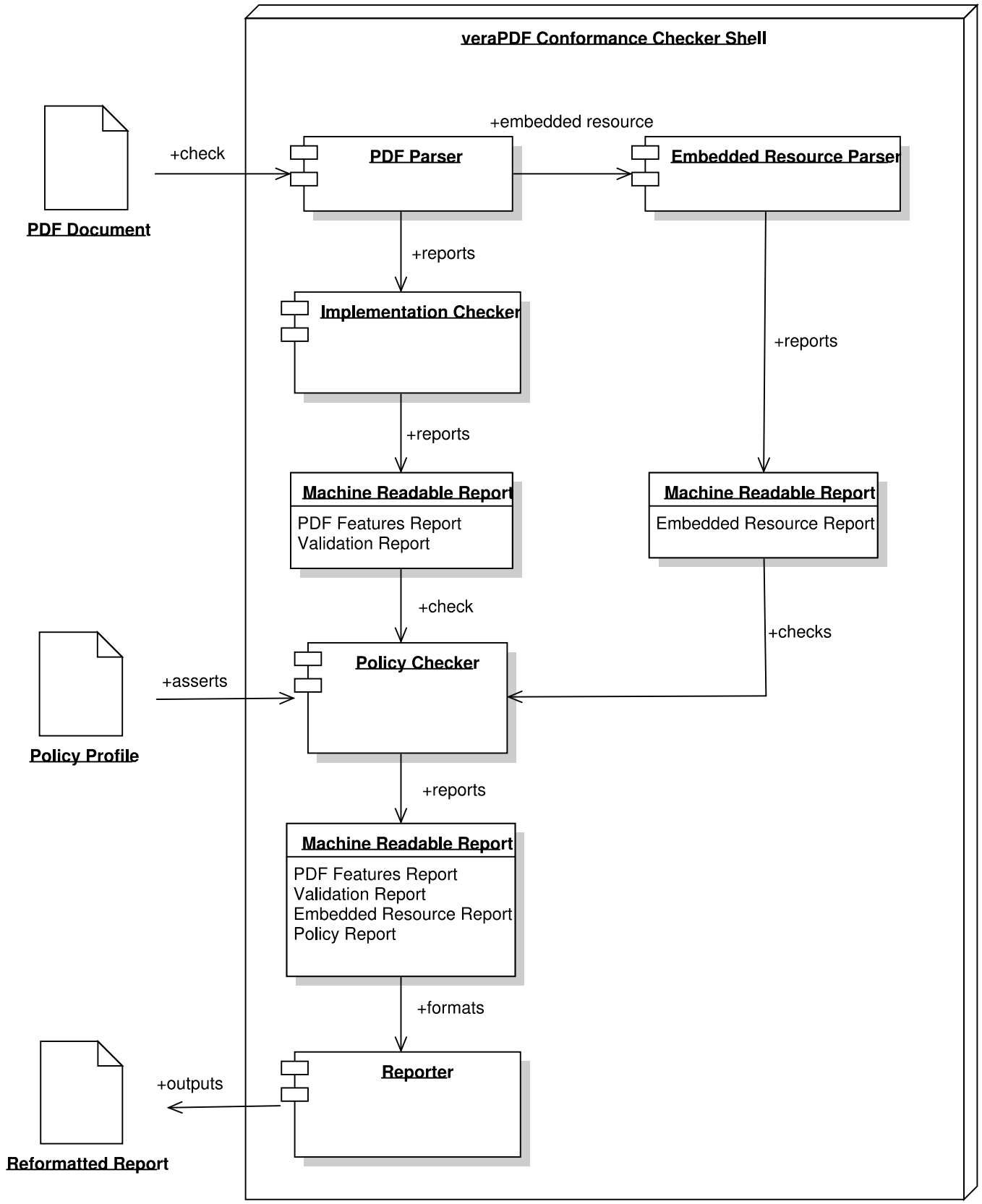
The PDF Parser parses a PDF Document and provides a PDF Document Extract to the Implementation Checker (which uses it to generate a PDF Features Report) and extracts Embedded Resources (such as images, fonts, colour profiles, and attachments) which may claim to be encoded in a format specified by external standards (as described in FS 1.2.1 PDF/A requirements beyond PDF syntax) and passes them to an Embedded Resource Parser (specialist parsers or validators for formats other than PDF/A).

Embedded Resource Parsers can be integrated using the interface defined in TS 9 Integration with third-party tools. Note that this requires technical skills and is not part of the default behaviour of the Conformance Checker. Integration of an Embedded Resource Parser registers the formats it can handle with the Conformance Checker by Mime-type. The PDF Document Extract (and PDF Features Report) identifies the Mime-type of Embedded Resources as described TS 4.2.4.5 Embedded files. If Embedded Format Parsers are available to handle the identified formats then the Embedded Resources are passed to an Embedded Resource Parser which returns an Embedded Resource Report which is available to the Policy Checker.

Policy Profiles can include rules to check against the Embedded Format Report. In this way, additional Policy Checks can be defined which cover detailed aspects of the Embedded Resources, such as their conformance to external standards, which can be reported to the user in Policy Reports.

Note that in this context that the Conformance Checker is reliant on the quality of the Embedded Resource Parser. No claims are made that using this mechanism will result in “definitive” validation as we define it for PDF/A in the veraPDF Conformance Checker (specifically the Implementation Checker).

FS 3.1.3 Functional architecture



FS 3.2 Integrations with other software

The original veraPDF Tender Proposal, section 1.1 III *Combinations with other software* (pp. 12-13) proposes integrations with Archivematica and DSpace. In addition to these integrations, which will still be carried out, we are adding another tool - JHOVE - to the list.

FS 3.2.1 JHOVE

Since our proposal for Phase 1 was submitted, the Open Preservation Foundation has taken over stewardship of JHOVE. The results of our Community Survey (due for publication in March 2015) will demonstrate that JHOVE is amongst the most widely adopted digital preservation tools available today.

JHOVE provides a PDF module for validating PDF Documents which has known issues as identified at the Open Preservation Foundation event Preserving PDF: Identify, Validate, Repair. veraPDF will carry out two activities relating to JHOVE, aligning with our ongoing maintenance efforts:

- integrate the veraPDF Conformance Checker as a new module in JHOVE dedicated to PDF/A Validation. This will immediately be available for the entire user-base of JHOVE;
- carry out a roadmapping exercise, aligned with the longer-term ambitions for veraPDF as a general PDF Validator (as described in the original veraPDF Tender Proposal section II *Potential of the Proposed Idea/ Solution/ Technology to Address Future and/ or Wider Challenges in the Area*) to identify how the PDF module within JHOVE may be replaced. Note that this is a longer-term ambition reliant on on-going stewardship of both JHOVE and veraPDF by the Open Preservation Foundation and will not form part of the work funded by PREFORMA.

In additional to stand-alone use, JHOVE is also integrated into both Preservica and Rosetta (commercial digital preservation systems). By integrating veraPDF into JHOVE the Conformance Checker will also be made available to all customers and users of those systems.

FS 4 Interfaces

The veraPDF Shell provides several user interfaces to the Conformance Checker designed for users with varying use cases or requirements and varying levels of expertise (from non-expert to expert).

Our approach to interface design will apply the principles and approaches of User Experience Design (UXD) including:

- usability (based on persona profiles derived from the stakeholder analysis and user research as described in CE 1 Stakeholders and CE 2.2 Specific communities);
- information architecture (based on the data available to the Conformance Checker as described in TS 4 Machine-readable Report format, and TS 8 Report Template format);
- interaction design (including wireframe prototyping for graphical user interfaces);
- visual design (including branding, look and feel, graphics).

See also TS 8.2 Accessibility for a description of the accessibility standards which will apply to all Graphical User Interfaces and Human-Readable Reports presented through the user interfaces.

FS 4.1 Standalone Distribution

A single-user installation intended for laptops, desktop PCs, and workstations which provides two Shell interfaces. The distribution will be made available as software packages prepared for single click install on Linux, Mac OSX, and Windows machines. The standalone distribution will package all dependencies required for operation, even in a non-networked environment (see TS 1.7.3.1 Standalone for details).

FS 4.1.1 Command Line Interface (CLI)

The Command Line Interface provides access to all the functional components of the Conformance Checker and allows the user to control them individually or separately. The expectation is that the Command Line Interface will require familiarity with the principles and approaches of format validation, metadata generation and transformation, and use of the terminal in their chosen operating system.

Examples of Command Line Interface usage are provided in FS 4.3 Command Line Interface examples.

Target audience	Expert users, sysadmins, developers
Functional principles	The full range of Conformance Checking functionality should be available.

FS 4.1.2 Desktop Graphical User Interface (GUI-D)

The Desktop GUI provides ‘point-and-click’ access to predefined sequences of operation of the Conformance Checker components. For example to generate reports from a directory of files or apply Policy Profiles obtained from the Policy Profile Registry. The Desktop GUI will enable the easy selection of configuration options including PDF/A Flavours for Validation Profiles, Policy Profiles provided for common policy requirements identified from the community, turning Metadata Fixing on or off, or outputting reports in common formats using supplied Report Templates.

Target audience	Non-expert users
Functional principles	Simplified options should be available to provide quick access to commonly used functionality through the loading of predefined configuration parameters (e.g. input/output locations, Policy Profiles, Report Templates).

FS 4.2 Server Distribution

A server installation which will require some technical expertise to install and configure, typically that of a sysadmin, which provides one Shell interface. The distribution will be made available as software packages prepared for straightforward install on common server platforms including Debian/Ubuntu, Fedora, and Suse. The server distribution will package all dependencies required for operation (see TS 1.7.3.3 World Wide Web).

FS 4.2.1 Web Graphical User Interface (GUI-W)

The Web GUI is comparable in intent to the Desktop GUI but provided for use in a networked environment where the user interacts with a website using a browser. As with the Desktop GUI, the Web GUI will enable the easy selection of configuration options (see 4.1.2 Desktop Graphical User Interface). The Web GUI is bult on the Web REST API (see TS 1.6.3.5 REST API).

Target audience	Non-expert users
Functional principles	Simplified options should be available to provide quick access to commonly used functionality through the loading of predefined configuration parameters (e.g. input/output locations, Policy Profiles, Report Templates).

FS 4.3 Command Line Interface examples

This section provides examples of using the Command Line Interface to operate the Conformance Checker to meet selected use cases. Note that only a few examples are provided to illustrate the functionality. All Command Line Interface parameters will be specified fully in technical documentation released with the Conformance Checker.

FS 4.3.1 Implementation Checker and Metadata Fixer

This example shows how the Command Line Interface would be used to meet the following use cases:

- FS 2.1.1.1 Generate a PDF Features Report;
- FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour;
- FS 2.2.1.2 Fix PDF Metadata and produce a new PDF Document;
- FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy, Metadata Fixing)

FS 4.3.1.1 Input

Input or configuration	Description
PDF Document	Memory, file, http(s), (s)ftp, Cloud
Validation Profile	Profile that corresponds to a specific PDF/A Flavor
Option to fix metadata	Allows modifications of the input PDF Document
Option to include PDF Features Report in the Machine-readable Report	Enables optional inclusion of the PDF Features Report

Input or configuration	Description
Verbosity level of the PDF Features Report	Defines which information should be included into the PDF Features Report NOTE: Report Templates defining fixed verbosity levels will be provided, users may also supply their own.
Progress reporting	The way to report the progress. In case of the API this is a callback passed into validation classes.
Option to stop after N errors	Allows interrupting the validation
Temp directory	There is the default value, but custom temp folder may be required because of a need to: ● provide a larger filesystem volume; ● conform to access restrictions.

FS 4.3.1.2 Output

Output	Description
Repaired PDF Document	Memory, file, http(s), (s)ftp
Machine-readable Report (containing the PDF Features Report, Validation Report, and Metadata Fixing Report)	Memory, file, API object
Progress reporting	The progress is reported according to the corresponding input parameter. For example, in case of veraPDF Command Line Interface it may be converted to text output to stdout
Exit code	OK or error code indicating that the software was not able to complete the task In case of an error code the error details may be logged additionally to the standard output stream (English only).

FS 4.3.1.3 Parameters

Parameter	Description
-validate (-file <filepath>|-url <URL>)	[Required] Tells CLI to perform validation and defines input PDF Document The parameter **-file** is followed by an absolute local or UNC file path of the PDF to verify The parameter **-url** is followed by a URL (in URI-encoded form) that can be used to get the PDF Document to validate. The schema (protocol) and all the additional required information like server port, username, password etc must be included into the URL. The following schemas are supported: http/https/ftp/sftp and file (which means local file)
-pdfa (none|1a|1b|2a|2b|3a|3b|3u)	[Required] Defines the PDF/A Flavor to be used to validate the input PDF Document The value **none** may be used to skip PDF/A Validation in case only the PDF Features Report is required.
-fixmetadata	[Optional] If present tells the CLI to automatically fix (if possible) PDF Metadata so that it is compliant with the requested PDF/A Flavor. All the successful and unsuccessful fixes are logged in the Metadata Fixing Report.
-verbosity [0-9|-file <filepath>]	[Optional] If present tells the CLI to add the PDF Features Report (including PDF Metadata and all other XMP packages available in the PDF Document) into the Machine-readable Report generated according to **-pdfa** or **-preflight** option. The combined report is handled as specified by the **-report** parameter The extra parameter defines verbosity level (i.e. which information that should be included) of the PDF Features Report. If the option is missing the default value “3” is used. The parameter **-file** is followed by an absolute local or UNC file path of an alternative Report Template. The Template can be used, for example, to fine-tune the verbosity for specific aspects of the PDF Features Report or transform to other formats.

Parameter	Description
-progress (stdout|-file <filepath>|-url <URL>)	[Optional] Defines the destination for the progress reporting. If the option is missing no progress is logged The parameter **stdout** means the progress information will be written to stdout in a way similar to the way used for **-file** option The parameter **-file** is followed by an absolute local or UNC file path of a writable file. The CLI will append the progress information at the end of this file. The file should be seekable so a client software can open it in read-only mode and periodically read the new progress information from the end of this file. Each progress record is the textual representation of an integer number from 0 to 100 indicating a percentage of the completion; each record is written in a new line The parameter **-url** is followed by a http/https URL (in URI-encoded form) that can be used to send POST requests containing the progress information
-stoperrors <number>	[Optional] Defines the number of failed Validation Checks after which the process is interrupted. If this parameter is missing the validation will be performed completely disregarding the number of detected errors The value must be an integer number greater than 0
-tempdir <folderpath>	[Optional] Defines the path to the temp folder that will be used by the CLI to store temporary data. If this parameter is missing the CLI will use the temp folder provided by Operating System (normally current user temp folder) The parameter **-tempdir** is followed by an absolute local or UNC folder path that points to a writable folder on a volume with enough space for CLI to proceed NOTE: CLI will clean up all the temp files it creates but in case of forcible termination some files may remain and have to be cleaned up manually

Parameter	Description
-output (-file <filepath>|-url <URL>|inputpath)	[Required if **-fixmetadata** option is used] Defines output PDF stream The parameter **-file** is followed by an absolute local or UNC file path to a writable location that shall be used to save the modified PDF. If there is no modifications made the original PDF Document will be saved. If there is an existing file at this location it will be overwritten. The output path can be the same as was used for input The parameter **-url** is followed by an URL (in URI-encoded form) that can be used to upload the new PDF Document. The schema (protocol) and all the additional required information like server port, username, password etc must be included into the URL. The following schemas are supported: http/https/ftp/sftp and file (which means local file) The value **inputpath** can be used in case the input PDF is provided via **-file** parameter and means the output PDF Document shall overwrite the input PDF Document.
-report <filepath>	[Required] Defines the path to save the Machine-readable Report The option **-report** is followed by an absolute local or UNC file path to a writable location that shall be used to save the report. If there is an existing file at this location it will be overwritten The report will include the Validation Report (if **-pdfa** is not **none**), the PDF Features Report including all the document metadata (if the parameter **-details** is used), and the Metadata Fixing Report (if the parameter **-fixmetadata** is used)

FS 4.3.1.4 Invocation

```
verapdf -validate -file C:\test.pdf -pdfa 1a -fixmetadata -verbosity 9 -
progress stdout -stoperrors 1 -tempdir C:\Temp -output -file C:\test_fixed.pdf
-report C:\test_report.xml
```

*This is the command to execute CLI(program with the name **verapdf**) validation for the input file test.pdf, validate on PDF/A-1 level a with automatic metadata fixing. In addition to performing regular PDF/A validation CLI will also report PDF Features with the verbosity level 9. The progress will be reported in console (stdout). The validation must stop as soon as at least 1 error is encountered. The temp folder is C:\Temp. The new PDF will be saved into the file test_fixed.pdf, the Report into the file test_report.xml.*

```
verapdf -validate -url https://verapdf.com/pdfs/test.pdf -pdfa 1b -progress -
url https://verapdf.com/progressreporter/ -output -url
ftp://verapdf.com/verifiedpdfs/test_fixed.pdf -report C:\test_report.xml
```

In this example the input file is defined by a URL, as well as the modified file and the destination for

progress reporting.

FS 4.3.2 Policy Checker

This example shows how the Command Line Interface would be used to meet the following use cases:

- FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements;
- FS 2.4.1.1 Obtain a Machine-readable Report (PDF Features, Validation, Policy, Metadata Fixing)

FS 4.3.2.1 Input

Input	Description
PDF Features Report	Memory, file, API object
Policy Profile	Memory, file, API object

FS 4.3.2.2 Output

Output	Description
Machine-readable Report (containing the Policy Report)	Memory, file
Exit code	OK or error code indicating that the software was not able to complete the task In case of an error code the error details may be logged additionally

FS 4.3.2.3 Parameters

Parameter	Description
-check <filepath>	[Required] Tells CLI to perform Policy Checks and defines input PDF Features Report The parameter **-check** is followed by an absolute local or UNC file path of the Machine-readable Report containing the PDF Features Report
-profile <filepath>	[Required] Defines the Policy Profile The parameter **-profile** is followed by an absolute local or UNC file path of the Policy Profile to be used for Policy Checking
-report <filepath>	[Required] Defines the path to save the resulting Policy Report The parameter **-report** is followed by an absolute local or UNC file path to a writable location that shall be used to save the Report. If there is an existing file at this location it will be overwritten.

FS 4.3.2.4 Invocation

```
verapdf -check C:\pdffeatures_report.xml -profile C:\policy_profile.xml -report
C:\policy_report.xml
```

This is the command to perform Policy Checks using the input Machine-readable PDF Features Report pdfdetails_report.xml. The Policy is defined by the Policy Profile policy_profile.xml. The resulting Policy Report will be saved to policy_report.xml.

FS 4.3.3 Reporter scenarios

This example shows how the Command Line Interface would be used to meet the following use case:

- FS 2.4.1.2 Obtain a Human-readable Report (PDF Features, Validation, Policy)

FS 4.3.3.1 Input

Input or configuration	Description
Machine-readable Report (any type)	Memory, file, API object
Report Template	The template to generate HTML or PDF Human-readable Report
Locale information	The Language Pack for the resulting Human-readable Report

FS 4.3.3.2 Output

Output	Description
Human-readable Report	Memory, file, HTTP response stream
Exit code	OK or error code indicating that the software was not able to complete the task In case of an error code the error details may be logged additionally

FS 4.3.3.3 Parameters

Parameter	Description
-convert <filepath>	[Required] Tells CLI to convert Machine-readable Report **<filepath>** into a Human-readable Report The parameter **-convert** is followed by an absolute local or UNC file path of a Machine-readable Report

Parameter	Description
-template <filepath>	[Required] Defines the Report Template for Human-readable Report generation The parameter **-template** is followed by an absolute local or UNC file path of the Template to be used for conversion
-language <language-code>	[Optional] Defines the language to be used for the resulting Report The parameter **-language** is followed by a language code that is defined as the combination: <primary-code>-<subcode> <primary-code> - two-letter code reserved for language abbreviations (ISO639) <subcode> - two-letter subcode that is a country code (ISO3166) Examples: en-US, de-DE If the option is missing the default language '**en-US**' is used
-report <path>	[Required] Defines the path to save the resulting Human-readable Report The parameter **-report** is followed by an absolute local or UNC file or folder path to a writable location that shall be used to save the Report. If there is an existing file/folder at this location it will be overwritten. The folder path is expected in case of HTML Report Template with the option to keep all the resources (CSS, JS, images) as external files

FS 4.3.3.4 Invocation

```
verapdf -convert C:\validation_report.xml -template C:\template.xslt -language de-DE -report C:\report.html
```

This is the command to convert the input Machine-readable Validation Report validation_report.xml to a Human-readable Report report.html. The Template is defined by the template.xslt and the Report language is German.

Technical Specification and Software Architecture

Supporting the Functional Specification, the Technical Specification defines the veraPDF Conformance Checker software, including the architecture and design, validation model, Validation and Policy Profile formats, Machine-readable Report format, Report Template format, integration with third-party tools, test framework, and internationalisation.

TS Summary of technologies

The veraPDF Conformance Checker will be built in Java.

Other technologies used are summarised in this table.

Technology	Application
PDF	reading, checking, modifying, writing, generation based on a template
XML	used for Machine-readable Reports, Policy Profiles
JSON	used as alternative machine readable form for web services and the like
XSLT,XPath	used in Policy Profiles and Human-readable Reports generation
XSL-FO	Human-readable PDF Reports generation
HTML, CSS, JS	Human-readable HTML Reports generation
XMP, XMP location path	reading, checking, modifying, writing
HTTP/HTTPS with REST	input/output PDF, progress reporting
FTP/SFTP/FTPS	input/output PDF
Fonts	checking embedded font programs in PDF
Image formats	JPEG, JPEG2000, TIFF, PNG etc (checking image resources in PDF)
ICC profiles	checking embedded ICC profiles in PDF
IPC	inter-process communication for the case of communication with an external executable such as a third-party plug-in image validation
TMX	for multi-language support in Human-readable Reports
Java internationalization	for shell messages and the like
Schematron	for expressing Policy restrictions

TS 1 Architecture and Design

This section describes the architecture and design of the veraPDF Conformance Checker. First the overall design principles are outlined, then an overview of the architecture and a domain model are presented. The remainder of the section gives details of the Java projects that make up the Conformance Checker.

TS 1.1 Design Principles

The architecture and design have been developed with three guiding principles in mind, simplicity, modularity, reliability, and use of open standards.

TS 1.1.1 Simplicity

The design has been kept as simple as possible. In general, measures of software reliability and maintainability decrease as a system's complexity increases. It's easier to specify the behaviour of small single-responsibility classes unambiguously, making them straightforward to implement. This also facilitates the development of unit tests, which benefit from clear specifications. Small, reliable classes provide the building blocks for complex behaviours and systems.

TS 1.1.2 Modularity

The design makes every effort to separate concerns so that modules perform logically discrete, well defined functions. Modules are designed to be independent and, where appropriate, interchangeable providing opportunities for reuse instead of repetition.

The architecture presented separates the Conformance Checker into three top-level modules which are then divided into packages and finally interface/class definitions. There are simple, clearly stated dependencies at each level of the design.

TS 1.1.3 Reliability

The Conformance Checker is intended for use by stakeholders with an interest in PDF reliability: memory institutions looking to safeguard the long term accessibility of digital material and PDF vendors looking to provide robust PDF editing software. We aim to provide these organisations with components that can be trusted to perform reliably in the long term. These aspirations are at odds with complex software that tries to provide diverse functionality.

Instead we've chosen to design simple, modular components that have deliberately limited functionality and need know as little about their external environment as possible. This aspires to the highest principles of software design best practice, valuing predictability and reliability over complexity.

TS 1.1.4 UML Diagramming Conventions

There are four stereotypes that are used with specific meaning in our UML diagrams. Our use is consistent with conventional use but that's not always well defined. UML is a flexible modelling framework applicable to Object Oriented Languages in general. These languages have their own idioms which inevitably make their way into a model. The first two definitions address code organisation and are straightforward:

- <<*module*>> represents a physical modules of logically or functionally related code. For a Maven built Java project such as ours these are equivalent to Maven Projects and their aggregated Maven Modules;
- <<*package*>> this has a specific Java meaning, where a package is collection of classes and interfaces organised by namespaces to prevent name clashes. Modules are composed of Java packages as well as other artefacts.

The other definitions are a specific to a view of the proposed system. We've divided the classes/objects into two types:

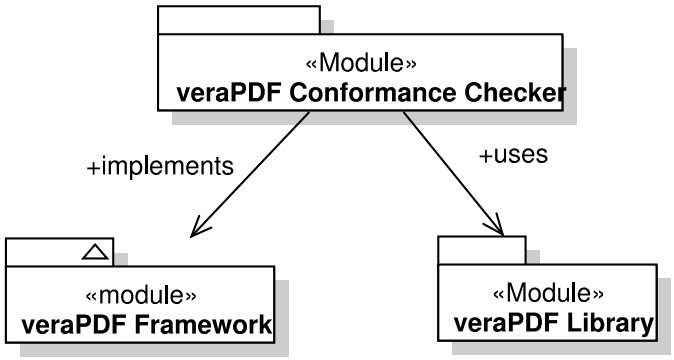
- *<<entities>>* entity classes model the information handled by the system, including Configuration, Validation and Policy Profiles, Reports and Report Templates;
- *<<interface>>* an interface defines a cohesive set of behaviours.

TS 1.2 Top level architecture

The veraPDF Conformance Checker design is divided into 3 top level modules:

- **veraPDF Library :** Java library that provides definitive Implementation Checking (PDF/A Validation and PDF Features Reporting) and Metadata Fixing for PDF Documents. The veraPDF library is designed for easy, adaptable access to PDF/A Validation, for use by developers and memory institutions with a deep interest in PDF;
- **veraPDF Framework :** Java library providing a definition and reference implementation of the Conformance Checker API, and a light framework to support developers implementing a Conformance Checker;
- **veraPDF Conformance Checker :** veraPDF implementation of a Conformance Checker combining functionality of the veraPDF Library with implementation of the veraPDF Framework**.**

These modules and their dependencies are shown below:



The veraPDF Conformance Checker can be compared to the tip of an iceberg. Although it sits at the top of the pile and delivers all of the working software it's really a thin wrapper around the APIs and functionality of the underlying veraPDF Library and veraPDF Framework. PDF-specific functionality will be added to the veraPDF Library while generic functionality will be implemented in the veraPDF Framework.

TS 1.3 Conformance Checker API

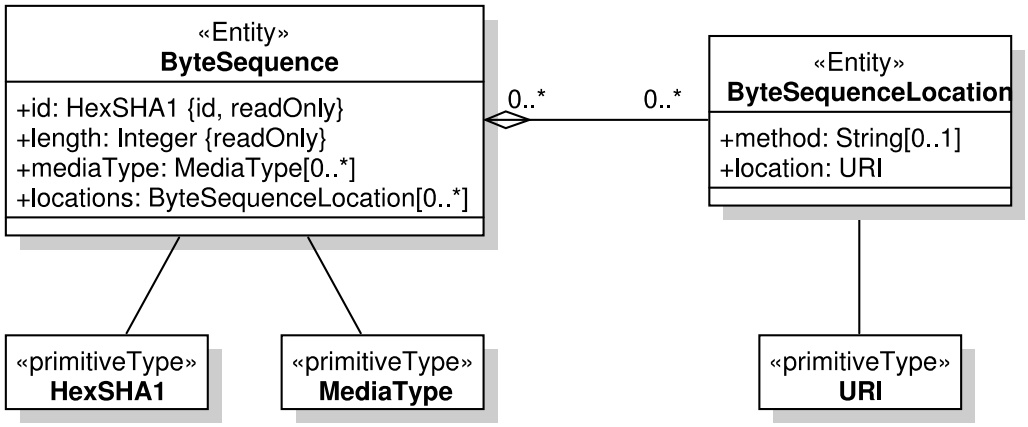
The Conformance Checker API is an abstract module that is defined independently of any implementation language. The veraPDF Framework is a Java realisation of this abstract module. This module describes the domain data types and entities operated on by the Conformance Checker components, i.e. Implementation Checker, Policy Checker, Metadata Fixer, Reporter, and Shell. The API is presented as an abstract UML model and is deliberately agnostic to:

- file formats checked, as assumed knowledge of specific format details could prevent interoperability between various conformance checker implementations;
- implementation, as using language specific constructs and data types hinders programmers developing conformance checkers in other languages; and
- deployment, thereby enabling deployment in the challenge brief scenarios and enhancing interoperability.

TS 1.4 Domain Model

The domain model comprises the main entities and interfaces. The model is presented in a series of UML diagrams, starting with low level entities which are, in turn, used to construct the Conformance Checker API.

TS 1.4.1 Primitive Types & ByteSequence Entities



TS 1.4.1.1 Primitive Types

The domain model defines three primitives beyond those commonly shared between language agnostic models (i.e. String, integer, etc.), they are HexSHA1, MediaType, and URI.

These types share three useful properties:

- they can be fully represented as a String value that's parsable according to an open standard;
- most development languages offer native libraries that support them, e.g. data types or functions to parse and use them;
- they each provide a well defined, standards-based identifier.

The three primitive types are described in the table below:

Primitive Type	Description
HexSHA1	The hexadecimal String representation of a SHA1 hash value. This is ALWAYS a string of EXACTLY 40 characters the form of which can be checked using the regex: `/^[0-9a-fA-F]{40}$/` The HexSHA1 value is useful as it uniquely identifies any Byte Sequence, within the probabilities of hash collisions. The value of it is deterministically derived from the contents of a specific Byte Sequence using an open, standard algorithm. By reading a Byte Sequence once you can verify that it's identical to any Byte Sequence that bears the same id. This means ANYONE can reliably create and compare the ids of Byte Sequences using an open standard.
MediaType	A Media Type is a string value the form of which is described in the Internet Media Type (MIME) RFC http://tools.ietf.org/html/rfc4288, see also http://tools.ietf.org/html/rfc2045, and http://tools.ietf.org/html/rfc2046. These underpin content type negotiation on the WWW and are used to indicate the type or format of a Byte Sequence, e.g. application/pdf.

Primitive Type	Description
URI	A Uniform Resource Identifier as defined by RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax. URIs provide resolvable identifiers that can be used to look up and obtain a particular resource either directly or indirectly through resolvers.

TS 1.4.1.2 ByteSequence

Refers unambiguously to a particular Byte Sequence and is an abstraction with only four properties.

Name	Type	Description
id	HexSHA1	The SHA1 hash value of the byte sequence which is its unique ID.
length	Integer	A whole number always where `length >= 0` This is obviously derived from the Byte Sequence by counting the number bytes. Used to let potential consumers know the size of the resource before reading.
mediaType	MediaType	This value is simply a possibly informed opinion at a point in time and not a deterministically derived or permanent property of the Byte Sequence.
locations	ByteSequenceLocation [0..*]	0 or more locations each of which provides an indication as to where a copy of the ByteSequence can be retrieved or written. The reader of the ByteSequence is responsible for re-calculating the SHA-1 value if they wish to guarantee the integrity of the read process.

The ByteSequence type has a simple default value for an empty document, expressed as XML it will be:

```
<byteSequence id=”da39a3ee5e6b4b0d3255bfef95601890afd80709” length=”0”
mediaType=”application/octet-stream”><locations></locations></byteSequence>
```

in JSON:

```
{
  "byteSequence": {
    "-id": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
    "-length": "0",
    "-mediaType": "application/octet-stream"
  }
}
```

TS 1.4.1.3 ByteSequenceLocation

Refers unambiguously to a particular Byte Sequence and is an abstraction with only four properties.

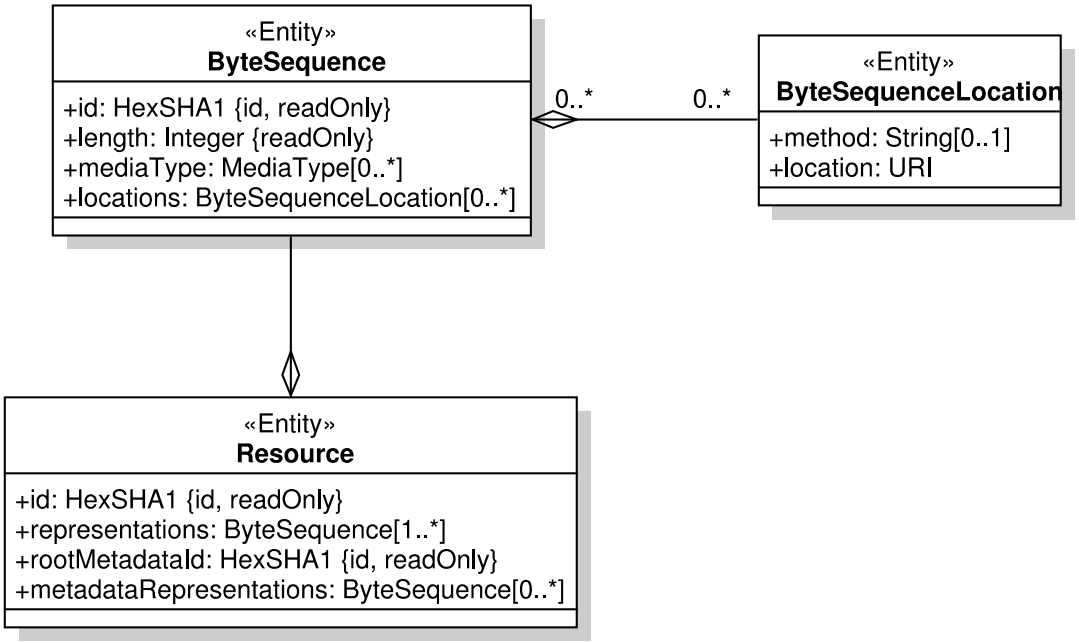
Name	Type	Description
method	String	A String value indicating HTTP method to use, indicates whether the resource is retrievable (GET), writable (POST, PUT), or to be removed (DELETE). Any location with an empty method string will be treated as a GET (read) location.
URI	location	A URI that provides either a URL or a resolvable URN that can be used to retrieve, write, or delete the ByteSequence.

A ByteSequence for a particular PDF Document will be:

```
{
"byteSequence": {
    "-id": "c433ee5e6b4b0d3255bfef95601890afd807094",
    "-length": "0",
    "-mediaType": "application/pdf",
    "locations": {
      "byteSequenceLocation": {
        "-method": "GET",
        "-reference": "http://verapdf.org/somefile.pdf"
      },
      "byteSequenceLocation": {
        "-method": "GET",
        "-reference": "file://localhost/home/username/somefile.pdf"
      }
    }
  }
}
```

To summarise, a ByteSequence is a Byte Sequence with built in identification and validation. A consumer knows the length of the object BEFORE they read it. It also has an indication of its type and a list of locations defining where it can be read and/or written.

TS 1.4.2 Resource Entity, Representations, and Metadata



A Resource is a new entity with important relationships. A Resource has the following attributes.

TS 1.4.2.1 Resource

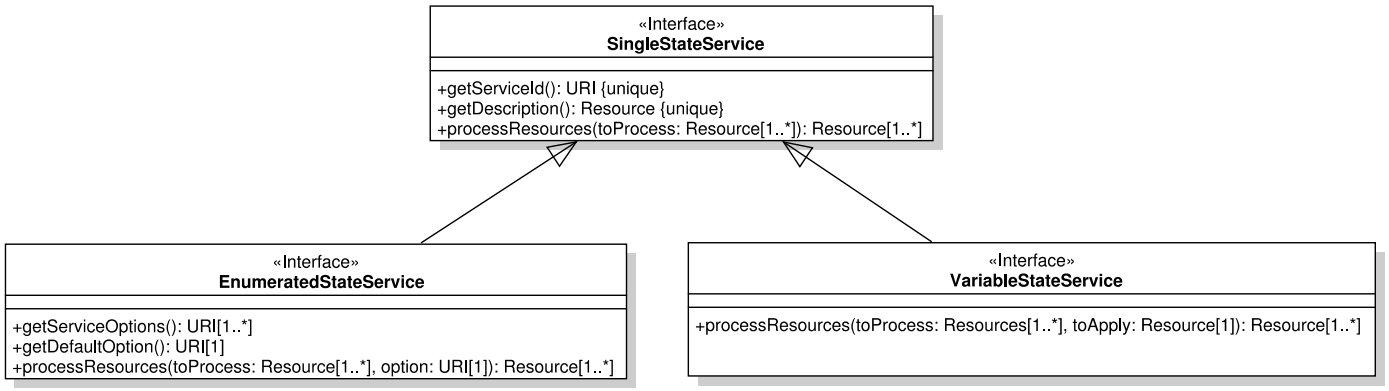
Name	Type	Description
resourceId	HexSHA1	The id of the Resource, this is generated from one of it's representations (explained shortly).
representations	ByteSequence [1..*]	1 or more ByteSequence that represent the Resource, there should ALWAYS be a Resource that matches the documentID, i.e. the same hash. Other entries are alternative representations of the Resource, the type of a representation is indicated by the ByteSequence MediaType. This is the identical Representation used in the term Representational State Transfer, i.e. RESTful Web Services.
metadataId	HexSHA1	The SHA-1 id of derived from one of the metadataRepresentations that could be considered a starting point for a Resource processor. This is only a convenience attribute and can be ignored by implementers. This is indicated by providing the empty hash string.
metadataRepresentations	ByteSequence[0..*]	0 or more ByteSequences that represent metadata, i.e data about the Resource.

TS 1.5 API Definition

This section defines the Conformance Checker API as a set of service interfaces. These interfaces do not correspond to the functional components that make up the Conformance Checker. Instead they define a set of interface behaviours that are flexible enough to implement them.

TS 1.5.1 Service Interfaces

The last elements of the Conformance Checker API are the service interfaces. At this level they do not have typed names that resemble the names of the Conformance Checker components. In describing the interfaces we show the relationships between them.



There are only three service interfaces: SingleStateService is a base interface which EnumeratedStateService and VariableStateService inherit from. Inheritance is an object-oriented construct but these interfaces could be implemented easily in a non object-oriented environment.

TS 1.5.1.1 SingleStateService

A single state service is designed to describe itself and perform a single, pre-defined function. The interface methods are:

Method	Description
getServiceId(): URI	This method takes no parameters and returns a unique URI that identifies the service. The URI can simply be used as a unique identifier, it doesn't have to be a resolvable location.
getDescription(): Resource	Another getter method with no parameters, this method returns a Resource that describes the service. The contents of the describing resource are up to the implementer. Development prototypes might return a Resource with some simple or non-existent descriptive information. A production ready, trusted service might contain a full description of the service, references to external standards documents that describe the service operation and a full change history of the service itself.
processResource(toProcess: Resource [1..*]): Resource	This method invokes the service operation on a set of Resources. The service should process a supplied representation of the Resource, e.g. a PDF Document, and add any results as either a metadata representation or representation held within the returned Resource.

This single state service is designed to perform one, self contained operation reliably. This could be calculating the SHA-1 hash of a Resource, or validating a PDF Document against a single PDF/A Flavour.

The limited options also limit the complexity of the service, which means it's easier to implement a reliable service, this reliability comes at the expense of flexibility.

TS 1.5.1.2 EnumeratedStateService

An EnumerateStateService is a specialisation of the SingleStateService designed to provide additional flexibility. It effectively allows the implementer to combine related SingleStateServices into a single service allowing the user to choose between them. In addition to those defined by it's parent interface an EnumeratedStateService offers the following methods:

Method	Description
getServiceOptions(): URI[1..*]	Returns a set of URIs that identify the options offered by the service. This allows a caller to establish the legal set of options supported by the service.
getDefaultOption(): URI[1]	Returns a single URI that identifies the service's default option, i.e. if no option is supplied by the caller the service will substitute the default option.

processResource(
  toProcess: Resource [1..*],
  option: URI[1]
): Resource
	This method invokes a service operation on a set of Resources.It differs from the processResource method of the SingleStateService in that it allows the user to supply a URI identifier that effectively chooses a processing option.

These services are more flexible than SingleStateServices. An example would be a service that supports three SHA hash algorithms, e.g. SHA1, SHA256, and SHA512. This service would offer three URI identifiers for its options, e.g.

- http://www.w3.org/2000/09/xmldsig#sha1
- http://www.w3.org/2000/09/xmldsig#sha256
- http://www.w3.org/2000/09/xmldsig#sha512

A different implementation might provide PDF/A Validation against an enumerated set of PDF/A Flavours and would thereby provide the functionality of the veraPDF Implementation Checker.

These services are more flexible than the SingleStateService but they are also more complex with a choice of execution paths.

TS 1.5.1.3 VariableStateService

VariableStateService is the most flexible service interface, allowing the user to pass an additional Resource alongside those to be processed. In the context of the veraPDF Conformance Checker this additional Resource might be a Policy Profile or a Report Template.

The single method is:

Method	Description

processResource(
  toProcess: Resource [1..*],
  toApply: Resource[1]
): Resource
	This method invokes a service operation on a set of Resources.The method requires an additional Resource supplied by the user that is used during processing.

The process is user-configurable and provides greater flexibility than the other service interfaces. While useful, the additional flexibility has implications for reliability and the ability of the implementer to reproduce

issues with the services operation as the execution of the service now depends on externally supplied input beyond the control of the implementer.

TS 1.6 veraPDF Framework

The entities and interfaces described in the domain model are deliberately abstract. This section describes a Java Framework that implements the API and provides a reference implementation of Conformance Checker components.

The veraPDF Framework is a Maven project divided into three sub-modules:

- **Conformance Checker API :** containing the domain model entities and service interfaces as Java interface definitions only. The interfaces are extended to provide further specialisations that model Conformance Checker components and types;
- **Framework Core :** provides implementations of the primitive data types, entities, and service interfaces. These are provided as concrete classes for data types, entities, and helper utilities with abstract classes providing defined points for extensibility, such as writing data to third party systems. This module also includes reference implementations of two general purpose components capable of:
    - making assertions about the presence or absence of patterns in XML trees (the Schematron Checker);
    - transforming XML documents into other XML documents, or other formats such as HTML, plain text, or XSL-DO (the Generic Reporter);
- **Shell Services :** provides reference implementations of Conformance Checker Shell services for the identification, storage, and retrieval of Byte Sequences and Resources.

The remainder of this section presents the different Java modules that make up the veraPDF Framework with descriptions of the key classes and interfaces.

TS 1.6.1 Conformance Checker API

A lightweight module that contains the Java interface definitions for entities and services from the domain model. These Java interfaces specify contracts for the behaviour of the classes that implement them. Their purpose is to allow an entirely greenfield implementation of the veraPDF Shell API without using the rest of the framework.

Interface based design

The API module is an example of a modular, Java Interface based design. This provides separation between the specification of behaviour and any implementation. The Framework Core module provides reference versions of Java classes that implement these interfaces. A developer could choose to write their own classes that implement these interfaces. These can be passed to any method that uses the Interface type rather than the implementation class.

Domain Model Types

The URI and MediaType primitives are provided by the JDK/JRE. URI is part of the Java networking library and MediaType is provided by the JAX-WS API so no type definitions are needed.

The HexSHA1 interface is defined with a single getValue() method that returns the hex String representation of the SHA1 hash. The entities and service interfaces are identical to those in the domain model with name changes that obey Java getter conventions for each of the properties described. We'll show these interfaces them when we cover the reference framework classes that implement them.

TS 1.6.2 Framework Core

Framework Core also contains reference implementations of these interface as concrete or abstract classes. Concrete classes are provided for the entity types and utilities Abstract classes are provided as extensibility points for integration with third party systems such as repositories and databases.

Not all classes are shown in detail. Main classes are shown but less functional classes, such as wrappers around collections of other data, are omitted (for example ComponentDetails which encapsulates the properties that uniquely identify a Component such as version number).

Use of Immutable Objects

Interfaces will not provide methods that can alter the state of an object after instantiation whenever possible. Mutable objects (i.e. exceptions to this approach) will be documented in code providing a justification for the decision. There are several motivations behind this practice:

- immutable objects are simpler to construct, test and use;
- truly immutable objects are always thread-safe;
- identity mutability is avoided;
- immutable objects have failure atomicity built in.

These properties of immutable objects support scalability and data integrity. Access to a mutable object from separate threads necessitates locking, reducing throughput and making code more difficult to maintain. Immutable objects eliminate this problem as their state cannot be changed so many threads can access a particular instance. Unchanging state provides data integrity as object instances and their properties cannot be accidentally altered.

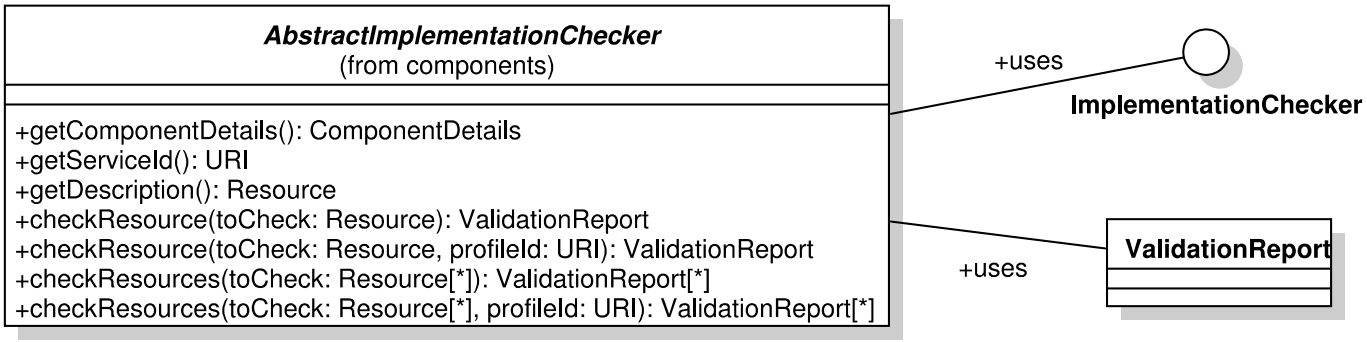
Favour composition over inheritance

Inheritance and polymorphism are often misused to create brittle class structures. Inheritance is intended to represent an "is type of" relationship but is often used to represent a "contains a" relationship, which is actually composition. Inheritance is only used when it leads to a simpler or more elegant design. Abstract base types will be delivered to provide common behaviour for some standard types but developers are not required to use them. Developers may implement interface behaviour as they see fit.

TS 1.6.2.1 Implementation Checker

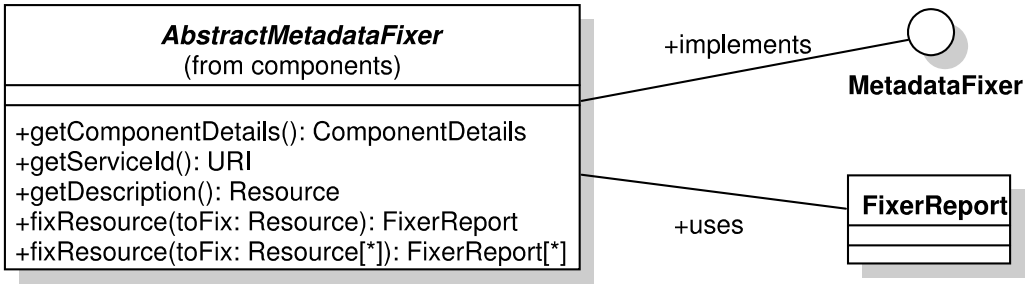
AbstractImplementationChecker provides reusable base class attributes and methods that deal with concerns common to all components such as id and description. This means that a developer extending the class does not have to implement the code and all components share a common implementation for these general concerns. The abstract class also provides conversion between types and methods for the Implementation Checker specialisation, Validation Report and checkResources, and the Resource type and EnumeratedStateService from the domain model. This allows the Java implementations to benefit from full type safety while still supporting the simple serialisable entities from the model.

The Implementation Checker interface also adds convenience methods for processing a single resource without the need to create a List with a single element. A developer extending the class must implement a checkResource method that provides a particular Implementation Check, for example PDF/A Validation.



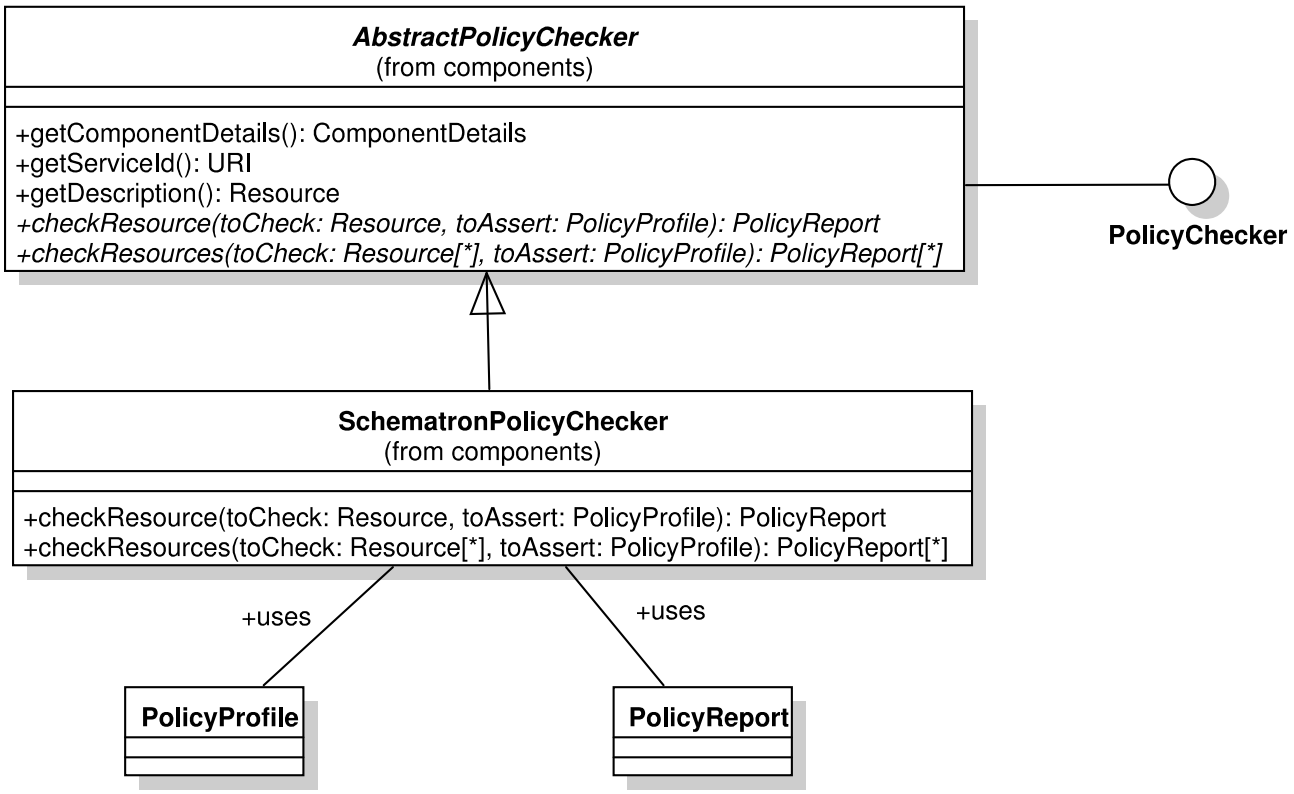
TS 1.6.2.2 Metadata Fixer

AbstractMetadataFixer provides similar support to that described for the Implementation Checker. Indeed the class shares its implementation of common component functionality with the Implementation Checker. Developers wishing to produce their own Metadata Fixer may concentrate on format and implementation specifics.



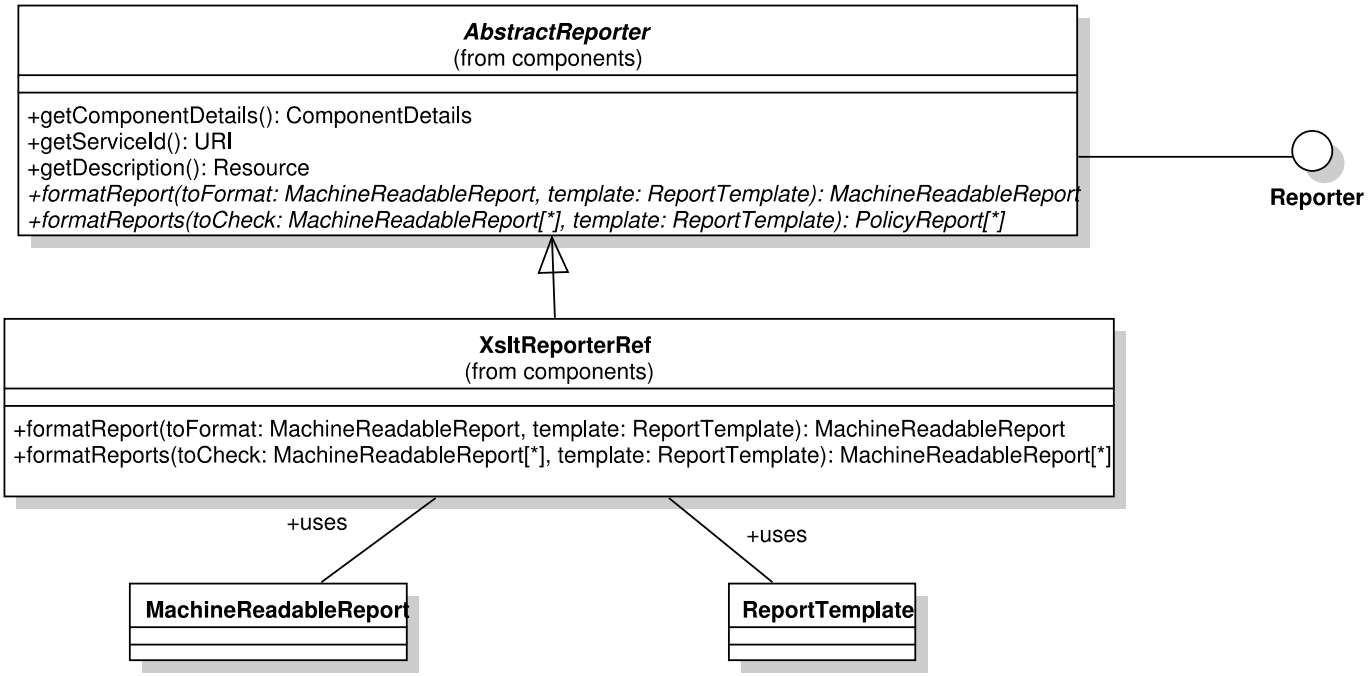
TS 1.6.2.3 Policy Checker

The Framework provides an Abstract Policy Checker similar to the components described previously. It also provides a generic Policy Checker implementation that applies a Policy Profile expressed as Schematron to a Resource that is expected to be an XML tree. The class can be used as provided for Schematron based checking, the only requirements are the availability of user defined Schematron Policy Profiles.



TS 1.6.2.4 Reporter

The Framework also provides a generic Reporter implementation as well as an abstract base class. The XSLT Reporter implements the Reporter interface and implements the formatReport and formatReports methods. This Reporter uses the supplied ReportTemplate as an XSLT transform and applies it to the passed MachineReadableReports which should be in XML format for compatibility with XSLT.



TS 1.6.3 ByteSequence & Resource Helpers

The Framework provides implementations of the ByteSequence and Resource types. Static factory methods to create ByteSequences and Resources from files, URLs, and InputStreams are provided to help developers create and work with these classes.

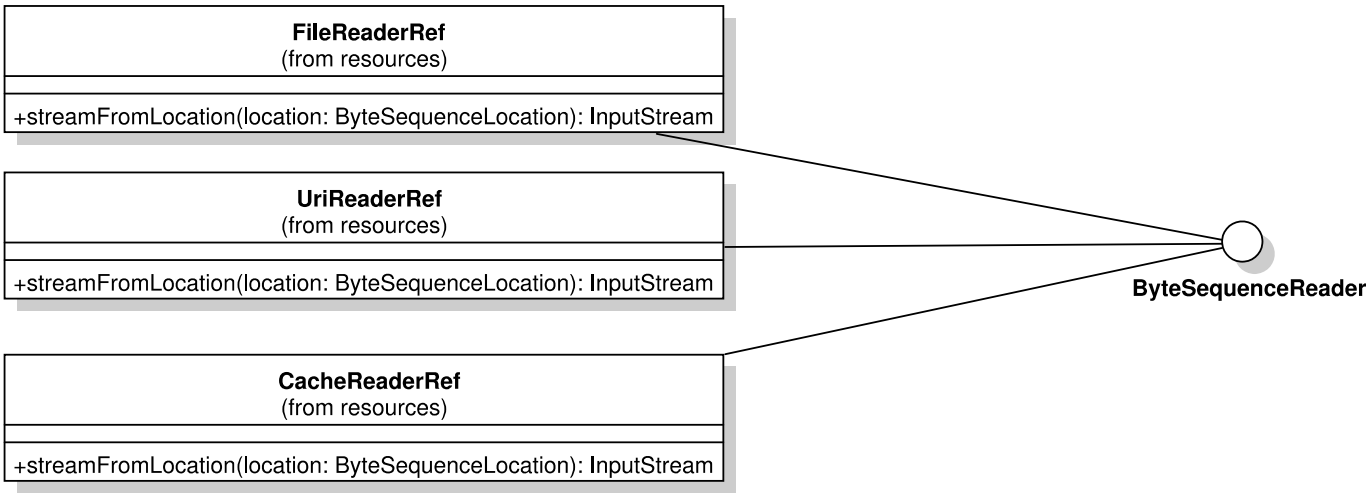
The Framework also provides ByteSequenceReader and ByteSequenceWriter interfaces. These interfaces can be implemented by developers wishing to integrate a Conformance Checker with an external system.

TS 1.6.3.1 ByteSequenceReaders

The Framework provides three reference ByteSequenceReader implementations:

- a file based reader that uses Java's native file I/O package;
- a URI reader based on Java's native networking package;
- a cache reader thats backed by memory for local caching.

Readers could be developed to read content and metadata from relational and document based databases, or repositories.

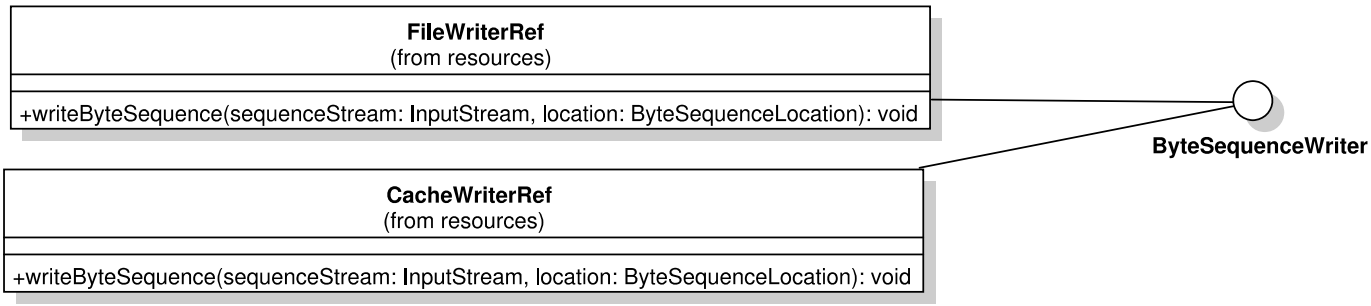


TS 1.6.3.2 ByteSequenceWriters

The Framework provides two reference ByteSequenceWriter implementations:

- a file based writer that uses Java's native file I/O package; and
- a cache writers thats backed by memory for local caching.

There is no standard for writing to a URL via HTTP, it takes place via a particular web service implementation. Writers can be developed that allow a Conformance Checker to write content or metadata to external systems.

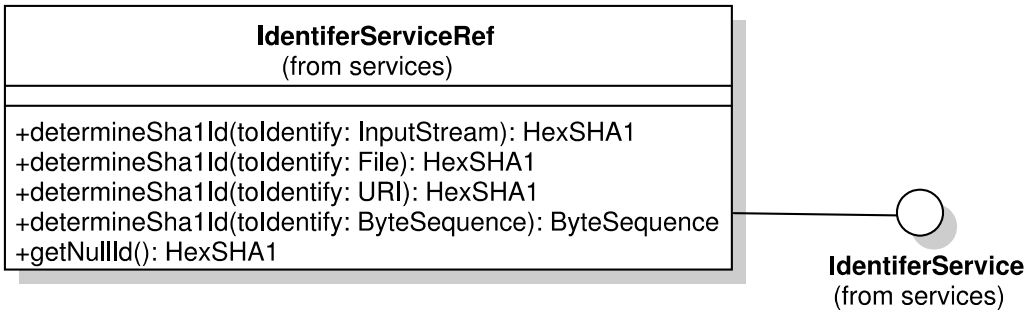


TS 1.6.4 Shell Services

A small set of utility services that provide core Shell functionality for use in Shell implementations.

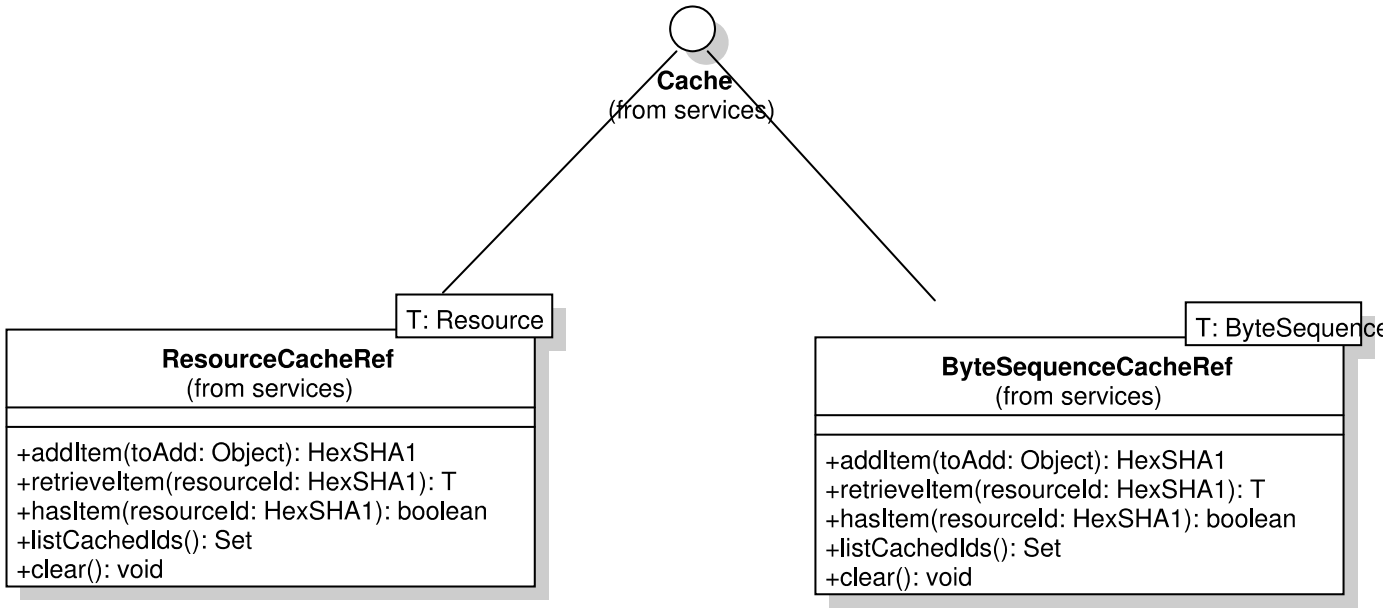
TS 1.6.4.1 Identifier Service

The Identifier Service is a specialisation of a SingleStateService. The service performs only one task: given File, Stream, or URL access to an unknown ByteSequence the service determines its SHA1 digest and returns a HexSHA1 String. The service can also be used as a hash checker if the SHA1 value is already known.



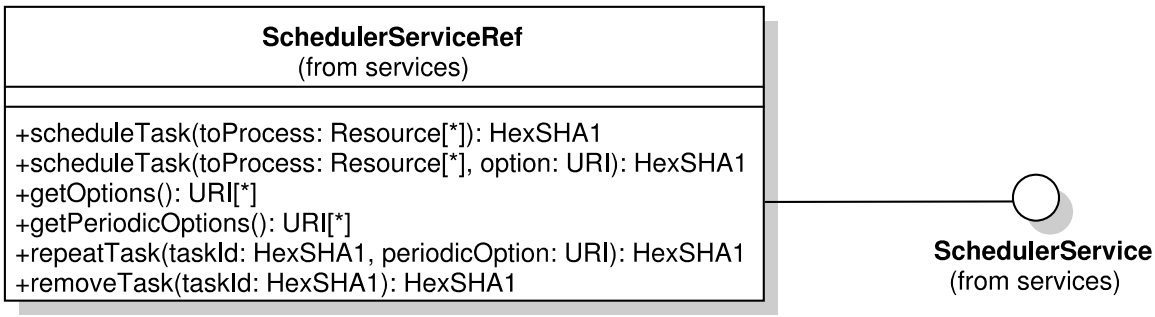
TS 1.6.4.2 Caching and Storage Services

The Framework provides reference caching and persistence services. These are all based upon a Cache generic interface and class that are enhanced by combining caches. The abstract Cache class can handle all types derived from a Resource, this includes specialisations such as Report Templates and Policy Profiles. Simple Cache types aren't persistent and lose their contents when the JVM terminates, they use either the temp directory or can use memory for performance. Persistent Stores are file backed and retain their state between executions.



TS 1.6.4.3 Scheduling Service

The Scheduling Service is an “always on” service that dispatches sequential, pre-configured component invocations that operate on a pre-defined set of Resources. The implementation is simple, a single thread sorts a set of tasks with the earliest on top. The thread then sleeps until the next task, but is also reawoken by the submission of a new task. Simple periodic repetition is supported, daily, weekly, monthly and annually.



TS 1.6.5 REST API

The veraPDF Conformance Checker Components will be deployable as REST web services. Note that REST APIs are not the same as programmatic, language dependent APIs. The REST web service mediates between requests received over HTTP and the underlying APIs and returns outputs from the APIs again over HTTP. This is illustrated using an example of an edited code snippet which defines a REST interface for the Identifier Service.

```
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

/**
 * REST resource definition for a ByteSequence identification service.
 * these are JAX-WS REST services and the annotations perform the magic of
 * handling content types and serialisation.
 *
 * @author <a href="mailto:carl@openpreservation.org">Carl Wilson</a>.</p>
 */
@Path("/identifier")
public class IdentifierResource {
    /**
     * @param uploadedInputStream
     *            InputStream for the uploaded ByteSequence
     * @param contentDispositionHeader
     *            extra info about the uploaded ByteSequence.
     * @return the {@link org.openpreservation.verapdf.HexSHA1} of
     *         the uploaded byte sequence serialised according to requested
     *         content type.
     */
    @POST
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML,
            MediaType.TEXT_XML })
    public HexSHA1 getHexSha1(
            @FormDataParam("file") final InputStream uploadedInputStream,
```

```
            @FormDataParam("file") final FormDataContentDisposition
contentDispositionHeader) {
        try {
            HexSHA1 id = ByteSequences.idFromStream(uploadedInputStream);
            uploadedInputStream.close();
            return id;// return
        } catch (IOException e) {
            // transfer fails, output exception and return empty stream id
            e.printStackTrace();
        }
        return ByteSequences.nullByteSequenceId();
    }

    /**
     * @return the {@link org.openpreservation.verapdf.HexSHA1} of
     *         an empty (0 byte) byte stream serialised according to
     *         requested content type.
     */
    @GET
    @Path("/null")
    @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML,
            MediaType.TEXT_XML })
    public HexSHA1 getEmptySha1() {
        return ByteSequences.nullByteSequenceId();
    }
}
```

The example uses the standard JAX-WS annotations to provide a REST endpoint:

- **@Path** defines the path of the resource, the URL location. These are normally stated relative to a master path provided by the web application. In this case we state that all services in this class are found at @Path(“/identifier”). If the containing web application provided a root path of http://verapdf.org/services then the identifier service endpoint would be http://verapdf.org/services/identifier.
- **@POST / @GET** defines the type of HTTP request that a particular method responds to. A single endpoint can service multiple request types.
- **@Consumes** indicates the type of content that an endpoint accepts. The getHexSha1() method in the example above declares `@Consumes(MediaType.MULTIPART_FORM_DATA)`. This is a MediaType (MIME Type) for multipart form data, a standard method of streaming binaries to a service.
- **@Produces** is used to advertise the type of content the service is capable of producing, in this case XML and JSON for both methods.

The web service supplies two endpoints:

- **.../identifier** allows a caller to POST (upload) a binary stream and returns the hex string value of the hash. If there’s a problem reading the file it returns the SHA1 hash of a null or empty stream. The caller can request that the service returns the digest value as either XML or JSON.
- **.../identifier/null** takes no parameter and simply returns the known constant for null stream SHA1 hash.

While interoperability between Conformance Checkers for different formats is out of scope during Phase 1 this REST web service design could provide an integration point for providing multiple Implementation Checkers behind a single endpoint and using content type negotiation to pass a submitted Resource to an Implementation Checker which can handle it.

For example, if a user passes a Resource using the application/pdf mediatype then the veraPDF Implementation Checker would be selected. Passing a Resource using the image/tiff mediatype would indicate that a TIFF Implementation Checker is required.

TS 1.7 veraPDF Conformance Checker

The veraPDF Conformance Checker module brings together definitive PDF/A Validation and Metadata Fixing from the veraPDF Library with the veraPDF Framework to deliver a fully functional Conformance Checker. This section gives a technical overview of how the Library and Framework are assembled.

TS 1.7.1 ByteSequences and Resources

The utilities provided by the Framework do not require additional coding to:

- create ByteStreams and Resources from PDF Documents as long they are available on a file system, a resolvable URI, or an open input stream;
- provide identifiers and data integrity through hash checking (as SHA1 support is built-in);
- persist ByteSequences and Resources (as caching and storage services are provided).

TS 1.7.2 Conformance Checker components

Conformance Checker components provide PDF-specific functionality.

TS 1.7.2.1 Implementation Checker

The abstract class provided by the Framework provides built in functionality to transform the incoming sets of Resources to individual ByteSequences to be checked. The implementer has the responsibility to:

- assign a URI identifier to their particular implementation and pass it to the abstract base class;
- assign a URI identifier to each of the enumerated service options (for PDF/A Validation this gives one URI for each PDF/A Flavour);
- write documentation to describe the service and include it as a Resource packaged in the components Java Archive (jar) file;
- populate the AbstractImplementationChecker internal URI options array, which will then return the available options if getServiceOptions() is called (the developer can select a default option, if they do not provide one the base class will automatically select one);
- ensure that the Validation Profile corresponding to a particular PDF/A Flavour is also selectable via the option URIs;
- call the veraPDF Library validation functionality, passing the PDF Document from the Resource and transforming the returned data for insertion into the returned Resource as metadata; and
- handle and report any exceptions thrown by the veraPDF Library, for example parse errors that occur when the passed Resource is not a PDF Document.

Note that the developer may give the class they derive from the abstract class any name they choose, within the constraints of Java syntax. The veraPDF derived class will be called VeraPdfImplementationChecker, but it can also be used as its interface type ImplementationChecker.

TS 1.7.2.2 Metadata Fixer

The implementer has similar responsibilities to those defined for the Implementation Checker. The developer must wrap the underlying veraPDF Library behind the MetadataFixer interface using the AbstractMetadataFixer base class. If the implemented Metadata Fixer is only required to write Repaired PDF Documents to a file system or local cache then the provided writer classes will suffice. Writing Repaired PDF Documents to external databases or systems would mean implementing a suitable ByteSequenceWriter.

TS 1.7.2.3 Policy Checker

The veraPDF Policy Checker uses Schematron Checks for properties in a PDF Feature Report via XPath, the XML query standard that underpins Schematron. The Framework provides a fully functional Schematron checker implementation so in this case no new classes or code are required. The developer is still responsible for providing a meaningful URI identifier for the new Policy Checker and other provenance information such as version numbering.

The developer must create and provide Policy Profiles capable of being applied to the PDF Features Report or Embedded Resource Report, as the Schematron checker is file format agnostic.

TS 1.7.2.4 Reporter

The veraPDF Reporter is simply an XSLT transforming service and is identical in function to the XsltReporterRef class. Beyond selecting an identifying URL and providing a service description a developer can simply use the XSLT reporter out of the box.

The developer must create and provide Report Templates to transform the format-specific data contained in Reports generated by the other components.

TS 1.7.3 Physical Architecture

In this section we consider the deployment scenarios presented in the PREFORMA Challenge Brief. We describe the features of our architecture and design that support each of the scenarios.

TS 1.7.3.1 Standalone

The veraPDF Conformance Checker components are designed to be deployable in standalone environments without access to network resources, including the Internet.

Each of the veraPDF components comes packaged with a rich Resource that:

- describes the function of the component;
- provides links to documents that cannot be included due to IPR (e.g. ISO specifications);
- includes provenance details of the component (e.g. details of the veraPDF consortium or a release changelog).

The Implementation Checker contains a tested and approved Validation Profile for each of the PDF/A Flavours it supports as a packaged resource in the Java archive (jar) file. The Metadata Fixer will encapsulate all supported fixes which can be performed without access to network resources.

The Shell is capable of storing user defined documents (Policy Profiles and Report Templates) and retrieving them from local disk storage using the provided storage services. It is possible that user defined Policy Profiles or Report Templates might access external resources (e.g. additional XML schema documents) that are beyond the knowledge or control of the implementer.

TS 1.7.3.2 Networked

Network deployment has built in support for networked file and URL resources through the supplied ByteSequence Readers and Writers, as described in TS 1.6.3.1 ByteSequenceReaders and TS 1.6.3.2 ByteSequenceWriters.

TS 1.7.3.3 World Wide Web

The REST API described in TS 1.6.3.5 REST API enables the deployment of the Components as web services. Deployment of the Java web service interfaces requires a standard Java servlet container. These could be deployed behind the same reverse-proxy server as the PREFORMA website, i.e. Apache or nginx web servers or on a different server, possibly at another URL.

Regardless of where the component services are hosted the PREFORMA website can provide a user interface that calls the service endpoint through a browser based client page written in HTML and Javascript. This could be a bespoke page in keeping with any design including the PREFORMA look and feel or a dedicated visual design provided by veraPDF.

TS 1.7.3.4 Legacy Systems

Integration into legacy systems requires the availability of a Java Virtual Machine within the legacy environment (i.e. hardware and operating system which supports it). If a JVM is available then the Conformance Checker can be deployed and executed alongside the legacy system. The APIs provide the integration points and Report Templates provide the mechanism for performing data conversions to formats compatible with the legacy system.

As described in the original veraPDF proposal and expanded upon in FS 3.2 Integrations with other software veraPDF will deliver integrations with Archivematica, DSpace, and JHOVE both to demonstrate the mechanism for integration with legacy systems which have different functional goals and to make the Conformance Checker available to users of those systems.

In the case where a legacy system runs on old or unusual hardware or an operating system that does not support a JVM direct integration is not possible. Instead, integration would rely on the development of a bespoke ResourceReader and/or ResourceWriter. The nature of any integration in this context would depend on specific detailed requirements for interoperability.

TS 1.7.3.5 DIRECT Evaluation Framework

The explanation of the DIRECT infrastructure provided by PREFORMA (suppliers' meeting 11/02/2015) describes the transfer of structured information, consistent with a pre-defined ontology, between a Conformance Checker and the evaluation framework (as opposed to the deployment of the Conformance Checker within the DIRECT operational environment). veraPDF will supply a Report Template that defines the transformation of our test results and performance metrics into the DIRECT ontology for use by the Reporter. The transformed data will be transferred to DIRECT via its submission interface.

TS 1.7.3.6 Scalability

Scalability of the Conformance Checker is addressed by making extra hardware resources available to the Conformance Checker. There are two vectors governing System scalability:

- vertical scaling: the process of adding resources to a server or desktop workstation (e.g. extra memory or a faster CPU);
- horizontal scaling: spreading the processing load across multiple CPU cores or machines.

Vertical scaling is the only solution for single threaded applications but high performance servers are extremely expensive, reflecting the high cost of single die, multi-core processors and high density memory chips. Horizontal scaling is generally more practical and affordable as commodity hardware provides more "bang per buck" in terms of computation power for the price. In order to allow horizontal scaling an application must be multi threaded, i.e. capable of running concurrently on multiple CPU cores or servers. See the results of the SCAPE project for more information about scalability in a cultural heritage context. During Phase 2 the Open Preservation Foundation will explore the possibility of using SCAPE technology under its stewardship to demonstrate the scalability of the Conformance Checker.

Objects that have been designed and tested with concurrent execution in mind are known as "thread-safe", meaning they're accessible by multiple threads simultaneously without clashes between threads. The immutable Conformance Checker classes and objects are designed with thread safety in mind (see TS 1.6.2 Use of Immutable Objects) which means that the veraPDF Conformance Checker can be deployed for horizontal scaling.

TS 2 Validation Model

The Validation Model responds to the basic research questions of PREFORMA: “how to interpret and implement standard specifications” and “how to determine whether a file is what it claims to be.” The model describes a generalised approach to file format validation which we apply to PDF/A specifically.

TS 2.1 Validation Model overview

The PDF Validation Mode is strongly motivated by Adobe Systems’ DVA model and whenever possible incorporates its basic building blocks. However, it is not so tightly linked to the PDF dictionary structure and defines the Validation Rules in a more general context suitable for defining Validation Profiles for other types of data structures. In particular, it may as well be used for validating ICC profiles, font programs and other additional specifications referenced by the PDF standard as described in FS 1 *PDF/A validation in context*.

We define the Validation Model based on the following concepts:

- **Object-oriented approach.** A tree-like hierarchy of object types that defines all possible types of objects we can check during PDF validation, their properties and inheritance rules.
- **Graph of associations.** The oriented graph of objects defines both the iteration rules and the list of checks per each object. Each edge in this graph may have additional marks, for example, specifying the number of objects of target type (0 or 1, 0 or more, 1 or more).
- **List of checks per each object type.** The checks themselves are defined as a single validation condition and an error/warning/info message it may result it. They follow the inheritance rules for the object types.
- **Validation conditions syntax.** A validation condition is a Boolean expression involving any object properties, global variables (see below), standard string and arithmetic expressions, and Boolean operators.
- **Variables.** There is a global storage of named variables accessible to all checks. Such global variables can be used in validation conditions (for example, the profile may include conditions for minimal/maximal page dimensions given as such variables). They can be also used to store intermediate values (calculated while performing some previous checks) relevant for further checks.

The above general principles of the Validation Model are technology agnostic and are not linked to any specific implementation framework and even to any serialization language.

TS 2.2 Terminology

Term	Definition
Document	Source file subject to certain specification for its internal format
Object	A logical piece of data within a given Document having certain Properties and Associated Objects
Object Type	A class of Objects with an identical set of Properties and Association Links.
Validation Rule	A condition imposed to all Objects of a certain Object Type

Term	Definition
Check	An act of verifying a given Validation Rule for a specific Object
Inheritance	A way to define a common set of Properties, Validation Rules and Association Links for classes of different Object Type
Inheritance Tree	A tree formed by all Object Types as nodes with oriented edges corresponding to their Inheritance
Object Property	A named property evaluated to one of the basic types such as boolean, integer, decimal, string
Associated Objects	Other objects related to a given Object according to the Document format
Association Link	A named link specifying that there is one or several Objects of Type A associated with all Objects of Type B
Objects Graph	A graph formed by all Object Types as nodes and all Association Links as oriented edges

TS 2.3 PDF Types Hierarchy

The Validation Model describes all Object Types, their Properties and their Inheritance information required to perform all checks of the given validation profile. Note that the object model does not need to cover all objects described in the PDF Specification.

This Validation Model is used for specifying the PDF Objects Hierarchy. If the validating tool does not support any of the Object Types or Association Links between them as specified in the Objects Graph the validation process is reported as failed. If the validating tool does not support any of the Object Properties used within the Rule, the corresponding check is reported as failed, but the overall validation process may continue.

All types form the tree with the type `Object` as a root. In other words, all types are inherited from `Object`, and each type except for `Object` has a unique parent type (i.e., there is no multiple inheritance). Below we list a sample model used for PDF/A validation. The complete PDF Types Hierarchy will be documented during Phase 2 and included into software documentation.

TS 2.3.1 Core types

```
Object -> CosObject, PDObject, Operator, External
```

TS 2.3.2 Cos types

```
CosObject->CosDocument, CosNull, CosBool, CosNumber (-> CosReal, CosInteger),
CosName, CosString, CosDict (->CosStream, CosFileDescriptor, CosTrailer),
CosArray
```

TS 2.3.3 PD types

PDObject->PDDocument, PDPage, PDResourceDict, PDResource, PDAnnot, PDAcroForm, PDAction, PDCMap, PDMetadata, PDGroup, PDOutputIntent*, PDAttachment

PDResource->PDFont, PDXObject, PDColorSpace, PDExtGState, PDPattern, PDShading, PDProperty

PDFont->PDType3Font, PDTrueTypeFont, PDType1Font

PDXObject->PDXForm, PDXImage

PDPattern->PDTilingPattern, PDShadingPattern

PDColorSpace->PDDeviceRGB, PDDeviceCMYK, PDDeviceGray, PDICCBased (->PDICCBasedGray, PDICCBasedRGB, PDICCBasedCMYK, PDICCBasedLab), PDLab, PDCalGray, PDCalRGB, PDSeparation, PDDeviceN

TS 2.3.4 Graphics operators model

Below we list all operators from ISO-32000:1 specification. Not all of them are required for the PDF/A validation.

Operator->OpGeneralGS, OpSpecialGS,OpPathConstruction,OpPathPaint, OpClip, OpTextObject, OpTextState, OpTextPosition,OpTextShow, OpType3Font, OpColor, OpShading, OpInlineImage, OpXObject, OpMarkedContent, OpCompatibility

OpGeneralGS->Op_w, Op_J, Op_j, Op_M, Op_d, Op_ri, Op_i, Op_gs

OpSpecialGS->Op_q, Op_Q, Op_cm

OpPathConstruction->Op_m, Op_l, Op_c, Op_v, Op_y, Op_h, Op_re

OpPathPaint->Op_s, Op_S, Op_f, Op_F, Op_f*, Op_B, Op_B*, Op_b, Op_b*, Op_n

OpClip->Op_W, Op_W*

OpTextObject->Op_ET, Op_BT

OpTextState->Op_Tc, Op_Tw, Op_Tz, Op_TL, Op_Tf, Op_Tr, Op_Ts

OpTextPosition->Op_Td, Op_TD, Op_Tm, Op_T*

OpTextShow->Op_Tj, Op_TJ, Op_', Op_"

OpType3Font->Op_d0, Op_d1

Op_Color->Op_CS, Op_cs, Op_SC, Op_SCN, Op_sc, Op_scn, Op_G, Op_g, Op_RG, Op_rg, Op_K, Op_k

OpShading->Op_sh

OpInlineImage->Op_BI, Op_ID, Op_EI

OpXObject->Op_Do

OpMarkedContent->Op_MP, Op_DP, Op_BMC, Op_BDC, Op_EMC

OpCompatibility->Op_BX, Op_EX

TS 2.3.5 External specifications

External->FontProgram, ICCProfile, CMapFile, ImageFile, XMPPackage, EmbeddedFile, Certificate

FontProgram ->Type1FontProgram, CFFFontProgram, TrueTypeFontProgram,

`OpenTypeFontProgram, CIDType0FontProgram, CIDType2FontProgram`

`ImageFile->JPEG2000, JPEG, JBig2, CCITT`

TS 2.4 Object Properties

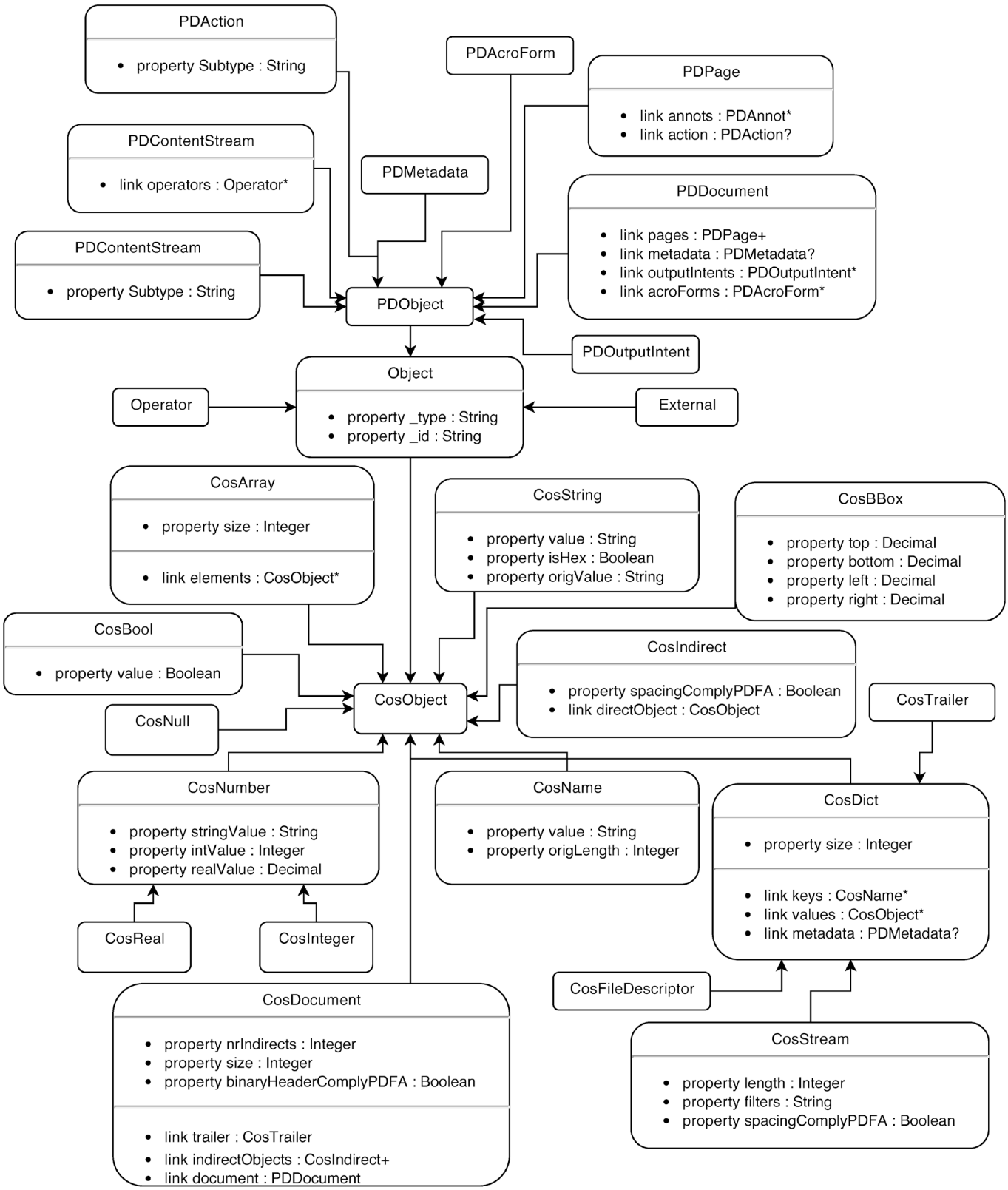
Each Object Type has a predefined list of Properties inheritable through the Types Hierarchy. Each property has one of the simple types such as `Boolean`, `String`, `Integer`, `Decimal` and may have a predefined value `null` meaning the property is not defined.

In case the object has an underlying PDF Dictionary structure and the property corresponds to a certain PDF key it shall use the key value as a name. However, the objects may have properties not directly linked to the PDF Dictionary structure. Evaluating such properties might involve complex logic, such as, for example, low level PDF/A requirements on PDF Document. A typical example would be a PDF/A clause that the PDF Document has a special binary header as a second line in the file. In the example below it is reflected via a special Boolean property `binaryHeaderCompliesPDFA` of the `CosDocument` object type.

TS 2.4.1 Examples of Properties

```
Object:
      type : String
CosDocument:
      nrIndirectObjects : Integer
      binaryHeaderCompliesPDFA: Boolean
PDDocument:
      nrPages : Integer
 PDXImage:
      Width : Integer
      Height : Integer
      BitsPerComponent : Integer
      Interpolate : Boolean
 ICCProfile:
      deviceClass : String
      colorSpace : String
```

A part of the above PDF Types Hierarchy is illustrated on the image below:



TS 2.5 Association Graph

Association Graph is an oriented marked graph where each node has one of the object types defined in the Types Hierarchy. The marks on the edge include:

- The name used for navigation through the graph. This name is identical to the PDF key if the object has underlying PDF Dictionary structure.
- The qualifier specifying the number min/max objects of the target objects. We commonly use ? (0 or 1), * (0 or more), + (1 or more) characters as shortcuts for this qualifier.

All edges starting at a given node form an ordered list that defines the validation order for the target nodes.

TS 2.5.1 Examples of Association Links

In this section, we list sample Association Links of the PDF Association Graph. The complete Association Graph will be specified during Phase 2 and included into software documentation.

```
CosDocument->CosObject+, PDDocument
PDDocument->PDAcroForm*, PDPage+, PDFormField*, PDMetadata, PDOutputIntent
Op_Do -> PDXObject
Op_Tf -> PDFont
Op_BDC -> PDProperty
Op_EI -> PDImage
PDImage -> PDColorSpace
CosDict -> PDMetadata?
PDICCBased->ICCProfile
```

TS 2.5.2 Validation Context

During validation process Objects of the same Object Type can be reached via different sequence of Association Links, or, in other words, via different graph paths. We call such path the **Validation Context**. It plays an important role and may be used to determine some Object Properties, which have special inheritance rules in PDF.

A typical example would be resource names not defined explicitly in the resource dictionary and defined via parent objects.

We always assume that the evaluation of Object Properties and all Checks take this Validation Context as one of the input data.

TS 2.6 Validation Rules

Each Validation Rule contains:

- **Metadata**: the unique ID, the description, reference(s) to the PDF/A specifications or other documents, severity (error, warning, info).
- **The test condition** evaluated to a boolean value. This test condition may refer to any Properties of the current Object, the values of global variables and may use the standard set of arithmetic, string, boolean operations
- **The message template** that can be inserted into this message. The template may use expressions %1, …, %9 as placeholders for the strings defined via expressions below
- Up to 9 expressions evaluated to string values and used for the message template placeholders
- Additional expressions used to set new values of global variable

TS 2.6.1 Examples of Validation Rules

Below we show examples of several Validation Rules described in a natural language. These rules are combined into the Validation Profile using the XML syntax. (See TS 3 Validation Profile format.)

Object type: `CosDocument`
Description: `checks if the second line is a comment starting with 4 binary characters`
Reference: `PDF/A-1 spec, 6.1.2 */`
Test condition: `binaryHeaderCompliesPDFA;`
Message template: `Missing binary comment after the file header`

Object type: `CosString`
Description: `the string is in Hex format and contains even number of raw bytes (before decoding)`
Reference: `PDF/A-1 spec, 6.1.6`
Test condition: `isHexFormat && rawLength % 2 = 0;`
Message template: `The hex string contains odd number of characters`

Object type: `CosInteger`
Description: `implementation limit`
Reference: `PDF/A-2 spec, 6.1.13`
Test condition: `value <= 2147483647 && value >= -2147483648`
Message template: `The integer exceeds implementation limits`

Object type: `PDDocument`
Description: `optional content is not allowed`
Reference: `PDF/A-1 spec, 6.1.13`
Test condition: `OCProperties == null`
Report message: `The document catalog dictionary contains /OCProperties key`

TS 2.6.2 Inheritance of Rules

If the Rules is defined for a base Object Type, it also applies to all Objects with the derived type. In such case, all Checks of the base Object Type should be performed prior to the Checks for the derived Type.

TS 2.6.3 Caching Check results

In most cases the same Object does not need to be checked more than once, even if it was reached via a different Validation Context.

However, there are cases when some Object Properties and, as a result, the related Checks depend on the Validation Context. A typical example would be validating the content of the Form XObject, which uses the resources not explicitly defined in the Form XObject resource dictionary.

So, any implementation of this Validation Model should property detect such cases and repeat the Check for the same Object only if the check does depend on the Validation Context. This would avoid entering into infinite loops during the validation process.

NOTE: There exist pathological cases in which, for example, the form object refers to itself via its name in the page resource dictionary. Proper implementation of the Validation Model should detect such cases, report an error and avoid infinite loops.

TS 2.7 Integration with third-party tools

A third-party tool may register Validation Rules for each Object Type. Whenever an Object of this Object Type is encountered, it is passed to the third-party tool for custom validation.

It is important to note that all such third-party checks shall still provide all required metadata and message template, so that the validation result can be embedded into the PDF Validation Report. However, they do not influence the final resolution, whether the document is compliant with a given validation profile.

TS 2.8 Validation algorithm

The above model uniquely defines the validation algorithm:

1. Start with a predefined root object (`CosDocument`)
2. Perform all Checks associated with its Object Type and with all Object Types it is derived from (eg., if the Object has type `CosStream`, then all checks for `CosDict` are also applied)
3. Retrieve all Objects associated with the current Object via all Association Links and perform step 2 for them.
4. Use *smart* caching mechanism to avoid checking the same Object twice.
5. Stop when there are no more Objects to check.
   NOTE. The order of all Checks is uniquely defined by the model and the validation profile. In particular, the reports generated by different implementations have to be identical.

TS 2.9 The formal syntax for the Validation Model

The Validation Model is serialized into a collection of text files following a custom veraPDF syntax resembling (but not identical) to an object-oriented programming language. It is defined in Wirth syntax notation as follows:

```
MODEL = (IMPORT)* (ENTITY)*
IMPORT = "import" QUALIFIED_NAME ";"
ENTITY = (COMMENT)* "type" QUALIFIED_NAME ("extends" SUPERTYPE)? "{"
            (ATTRIBUTE)* ";"
ATTRIBUTE = PROPERTY | LINK
PROPERTY = (COMMENT)* "property" QUALIFIED_NAME ":" TYPE ";"
TYPE = "String" |  "Integer" | "Decimal" | "Boolean"
LINK = (COMMENT)* "link" QUALIFIED_NAME ":" QUALIFIED_NAME ANY ";"
ANY = "?" | "+" | "*"
QUALIFIED_NAME = letter (letter | digit)*
COMMENT = "%" (character)*
```

Example

```
import org.verapdf.model.baselayer.Object;
import org.verapdf.model.pdlayer.PDDocument;

% Parent type for all basic PDF objects
type CosObject extends Object {
}

% Low-level PDF Document object
type CosDocument extends CosObject {
```

```
	% Number of indirect objects in the document
	property nrIndirects: Integer;
	% Size of the byte sequence representing the document
	property size: Integer;
	% trailer dictionary
	link trailer: CosTrailer;
	% all indirect objects referred from the xref table
	link indirectObjects: CosIndirect+;
	% true if the second line of the document is a comment with at
	% least 4 symbols in the code range 128-255 as required by
	% PDF/A standard
	property binaryHeaderCompliesPDFA: Boolean;
	% link to the high-level PDF Document structure
	link document: PDDocument;
}

% PDF Indirect object
type CosIndirect extends CosObject {
	% the direct contents of the indirect object
	link directObject: CosObject;
	% true if the words obj and endobj are surrounded by the
	% correct spacings accoring to PDF/A standard
	property spacingCompliesPDFA: Boolean;
}

% PDF Dict type
type CosDict extends CosObject {
	% number of key/value pairs in the dictionary
	property size: Integer;
	% all keys of the dictionary
	link keys: CosName*;
	% all values of the dictonary
	link values: CosObject*;
	% XMP metadata if it is present
	link metadata: PDMetadata?;
}
```

TS 3 Validation Profile format

The Validation Profile describes the Validation Checks which shall be applied during PDF/A Validation. Also it describes Metadata Fixes which may be performed in order to make the Document compliant with a Validation Check.

TS 3.1 Profile overview

The Profile strongly relies on the particular Validation Model that is described on the previous chapter. The Validation Model is expressed using a syntax described in TS 2.9 The formal syntax for the Validation Model. For different PDF/A Flavors veraPDF software may use slightly different Validation Models. So the Profile must have a reference to the specific Validation Model via the Model ID.

The Validation Profile format is XML.

TS 3.1.1 XML namespace and schema

The Profile namespace is defined by URI 'http://www.verapdf.org/ValidationProfile'.

The preferred prefix is 'vvp' (that is 'veraPDF Validation Profile').

The XML format supports Unicode so the Profile is able to contain any Unicode data. The Validation Profile normally uses UTF-8 encoding. The used encoding is anyway specified in the XML header.

TS 3.1.2 Text messages

All the text messages in the Profile are specified by string IDs which can be used to find the exact text for the message in a Language Pack.

TS 3.2 Profile structure

The Profile root element is **profile**. It contains the attribute **model** that provides the ID of the Validation Model this Profile was created for.

Example

```
<profile xmlns="http://www.verapdf.org/ValidationProfile"
model="org.verapdf.model.PDFA1a">
...
</profile>
```

The children of the root element are described in the table below

Element name	Presence	Description
name	once	The name of the Profile
description	once	The description of the Profile
creator	once	The creator of the Profile
created	once	The datetime when the Profile was created
hash	once	The SHA-1 hash code of the Profile NOTE: the hash is used to identify the Profile and it is generated based on the Profile Rules definition

Element name	Presence	Description
imports	once	Imports of other Validation Profiles
rules	once	Definitions of all the Validation Rules to be applied during validation

Example

```
<profile>
      <name>PDF/A-1a validation profile</name>
      <description>STR_ID_101</description>
      <creator>User1</creator>
      <created>2015-01-23T17:30:15Z</created>
      <hash>...</hash>
      <imports>
      ...
      </imports>
      <rules>
            <rule id="rule1">...</rule>
            <rule id="rule2">...</rule>
            ...
      </rules>
</profile>
```

TS 3.2.1 Rules

The element contains a list of child **rule** elements, each providing a definition of a specific Validation Rule.

Each **rule** element contains the attributes **id** and **object**. The object attribute is important for the validation algorithm as it allows identifying the validation model object to which the given Rule shall be applied.

Example

```
<rule id="rule1" object="CosDocument">
...
</rule>
<rule id="rule35" object="PDXObject">
...
</rule>
<rule id="rule112" object="PDAnnot">
...
</rule>
```

A definition of a Rule includes:

- description of the Rule
- test condition that is expressed according to the Validation Model
- error (or warning) message (optionally with arguments) that is issued if the condition is evaluated to false
- reference to the relevant specification and its clause
- optionally descriptions of the related Metadata Fixes

  NOTE: arguments may use the objects and their properties from the Validation Model to give more details about the problem

Element name	Presence	Description
description	once	The ID of the textual description of the Rule. The ID is later used for TMX localization Description example: “The % character of the file header shall occur at byte offset 0 of the file.”
test	once	The test condition expressed according to the Validation Model Example: "fileHeaderOffset == 0"
error or **warning**	once	The information that is added to the Machine-readable Report in case the test condition is evaluated to false This includes message and optionally arguments. The message is defined by its ID so it can be later used for TMX localization. Error message example: "Offset of the % character of the file header is %1 (note: value -1 means the file header is not found)." Argument: "fileHeaderOffset"
reference	once	The reference to the relevant specification and its clause. This element contains two children elements: **specification** - the specification name for which this rule was created **clause** - the number of the relevant clause in the specification
fix	none or more	The description of the Metadata Fix that may happen when applying this Rule

Example 1

The Rule for the CosDocument object.

```
<rule id="rule1" object="CosDocument">
      <description>STR_ID_401</description>
      <test>fileHeaderOffset == 0</test>
      <error>
            <message>STR_ID_402</message>
            <!--actual offset is the argument for the message-->
            <argument>fileHeaderOffset</argument>
      </error>
      <reference>
            <specification>ISO19005-1</specification>
            <clause>6.1.2</clause>
      </reference>
</rule>
```

In TMX file:

- STR_ID_401 is defined as the text: "The % character of the file header shall occur at byte offset 0 of the file."
- STR_ID_402 is defined as the text: "Offset of the % character of the file header is %1 (note: value -1 means the file header is not found)."

Example 2

The Rule for the PDAnnot object.

```
<rule id="rule112" object="PDAnnot">
	<description>STR_ID_570</description>
	<test>(F != null) &amp;&amp; (F_PrintFlag == 1) &amp;&amp; (F_HiddenFlag == 0)
	&amp;&amp; (F_InvisibleFlag == 0) &amp;&amp; (F_NoViewFlag == 0)</test>
	<error>
		<message>STR_ID_571</message>
		<!--actual flags values are the arguments for the message-->
		<argument>F_PrintFlag</argument>
		<argument>F_HiddenFlag</argument>
		<argument>F_InvisibleFlag</argument>
		<argument>F_NoViewFlag</argument>
	</error>
	<reference>
		<specification>ISO19005-1</specification>
		<clause>6.5.3</clause>
	</reference>
</rule>
```

In TMX file:

- STR_ID_570 is defined as the text: "An annotation dictionary shall contain the F key. The F key's Print flag bit shall be set to 1 and its Hidden, Invisible and NoView flag bits shall be set to 0."
- STR_ID_571 is defined as the text: "The F key in the annotation dictionary is not conforming: key is not present or its flags are not as required. Actual flags values: Print = %1, Hidden = %2, Invisible = %3, NoView = %4."

TS 3.2.1.1 Fix

The element provides a description of a Metadata Fix that can be performed when applying a rule. The applicable Metadata Fixes are a part of the Validation Model implementation. The description of a Fix consists of the elements in the table below.

Element name	Presence	Description
description	once	The ID of the textual string that represents the description of the fix
info	once	The message to be added into the Machine-readable Report in case the fix succeeded The element contains the child element **message** that is used to specify the ID of the actual message

Element name	Presence	Description
error	once	The message to be added into the Machine-readable Report in case the fix failed The element contains the child element **message** that is used to specify the ID of the actual message

Example

```
<rule id="rule53" object="PDMetadata">
      <description>STR_ID_608</description>
      <test>isInfoDictConsistent</test>
      <error>
            <message>STR_ID_609</message>
      </error>
      <reference>
            <specification>ISO19005-1</specification>
            <clause>6.7.3</clause>
      </reference>
      <fix id="fix1">
            <description>STR_ID_893</description>
            <!--the message in case the fix succeeded-->
            <info>
                  <message>STR_ID_894</message>
            </info>
            <!--the message in case the fix failed-->
            <error>
                  <message>STR_ID_895</message>
            </error>
      </fix>
</rule>
```

TS 3.3 Profile example

The self-documented example of the Validation Profile prototype: ProfileExample.xml

TS 4 Machine-readable Report format

TS 4.1 Report overview

The validation results are stored in the Machine-readable Report that is the XML file.

The Machine-readable Report may contain the following sections:

- general Document information
- processing information
- PDF Features Report
- PDF Validation Report
- Policy Report

The PDF Features Report contains the general information about the PDF Document, the description of the Document pages and resources. The amount of the information is controlled by a requested verbosity level (see FS 2.1.1.1 Generate a PDF Features Report). The PDF Features Report also contains all the XMP metadata packets in the original form. The PDF Features Report may also contain an Embedded Resource Report produced by an Embedded Resource Parser as described in FS 3.1.1.2 Embedded Resource Parsers.

The PDF Validation Report lists all the performed Checks and indicates those which revealed the violations of the PDF/A specification or other specifications it refers to (see FS 2.1.1.2 Check the conformance of a PDF Document to a PDF/A Flavour). The PDF/A Validation Report also contains the description of the performed Metadata Fixes (successful and failed) as described in FS 2.2 veraPDF Metadata Fixer.

The Policy Report provides the results of the performed Policy Checks (see FS 2.3.1.1 Check the conformance of a PDF Document to institutional policy requirements).

TS 4.1.1 XML namespace and schema

The Report namespace is defined by URI 'http://www.verapdf.org/MachineReadableReport'.

The preferred prefix is ‘vmrr’ (that is ‘veraPDF Machine-Readable Report).

The XML format supports Unicode so the generated Report is able to contain any Unicode data. The Machine-readable Report normally uses UTF-8 encoding. The used encoding is anyway specified in the XML header.

TS 4.1.2 Paths and URLs

The Report may contain file system paths. The paths are absolute and in platform-independent format which means the symbol ‘/’ is used as separator.

The Report may contain URIs. The URIs are URI-encoded.

TS 4.1.3 Text messages

All the text messages in the Report are specified by string IDs which can be used to find the exact text for the message in a Language Pack.

TS 4.2 Report structure

The Report root element is **report**. It contains attributes **creationDateTime** and **processingTime** which provide the time when Report was created and the time spent on the PDF Document processing respectively.

The children of the root element are described in the table below

Element name	Presence	Description
documentInfo	once	General information about the verified PDF Document
processingInfo	once	Environment, configuration information, and performance metrics for the performed Document validation task.
validationInfo	none or once	Information about the performed PDF/A Validation
pdfFeatures	none or once	PDF Document description including document, pages and resources details, metadata This element is present only if the PDF Features Report generation was requested
policyCheckingInfo	none or once	Information about the performed Policy Checks

NOTE: in case the Machine-readable Report is Policy Report instead of the **validationInfo** and **pdfFeatures** elements there is the element **policyCheckingInfo** that provides the information about the performed Policy Checks. The exact content of this element will be specified during the Phase 2.

Example

```
<report creationDateTime="2014-12-07T13:20:06.419+03:00" processingTime="00:00:02.319">
      <documentInfo>...</documentInfo>
      <processingInfo>...</processingInfo>
      <validationInfo>...</validationInfo>
      <pdfFeatures>...</pdfFeatures>
</report>
```

The sub-clauses below explain each element in more details.

TS 4.2.1 documentInfo

The children of the element provide the basic information about the processed PDF Document. The names of the elements:

- **fileName**
- **filePath**
- **size**
- **title**
- **author**
- **subject**
- **keywords**
- **creator**
- **producer**
- **creationDate**
- **modificationDate**
- **pdfVersion**

- **numOfPages**
- **maxPageSize**
- **tagged**
- **linearized**
- **encrypted**
- **trapped**
- **language**
- **hash**

Example

```
<documentInfo>
      <fileName>Test.pdf</fileName>
      <filePath>C:/Users/User/AppData/Local/Temp/Test.pdf</filePath>
      <size>1024000</size>
      <title>The document title</title>
      <author>The document author</author>
      <subject>The document subject</subject>
      <keywords>keyword1, keyword2</keywords>
      <creator>The document creator</creator>
      <producer>The document producer</producer>
      <creationDate>2014-11-23T15:41:28.018</creationDate>
      <modificationDate>2014-11-30T21:08:11.397</modificationDate>
      <pdfVersion>1.5</pdfVersion>
      <numOfPages>2</numOfPages>
      <maxPageSize width="210.001652" height="296.999959" unit="mm"/>
      <tagged>false</tagged>
      <linearized>true</linearized>
      <encrypted>true</encrypted>
      <trapped>unknown</trapped>
      <language>unknown</language>
      <hash>sha-1 hash code of the PDF document</hash>
</documentInfo>
```

NOTE: more details about the page boxes (Media/Crop/Trim/Bleed/Art boxes) as well as rotation and scaling factor for each page specifically can be found in PDF Features Report section.

NOTE: more details about the encryption and restrictions in PDF Features Report section.

TS 4.2.2 processingInfo

The children of the element are described in the table below

Element name	Presence	Description
installationConfig	once	Information about the software and the environment
taskConfig	once	Configuration configuration settings controlling software behaviour, these are reusable across executions and installations
executionConfig	once	Configuration settings unique to a particular validation task

Element name	Presence	Description
processMetrics	once	Provides information about the time taken to complete the task.

TS 4.2.2.1 installationConfig

The children of the element provide the basic information about the software and the environment. The names of the elements:

- **libraryVersion**
- **shellVersion**
- **javaVersion**
- **operatingSystem**
- **userName**
- **hostName**
- **tempDir**
- **homeDir**

Example

```
<installationConfig>
      <libraryVersion>1.0.2</libraryVersion>
      <shellVersion>2.0</shellVersion>
      <javaVersion>1.7.0_75</javaVersion>
      <operatingSystem>Microsoft Windows  Service Pack 1 (Build 7601)</operatingSystem>
      <userName>TestUser</userName>
      <hostName>Host1</hostName>
      <tempDir>C:/Users/TestUser/AppData/Local/Temp</tempDir>
      <homeDir>C:/Users/TestUser</homeDir>
</installationConfig>
```

TS 4.2.2.2 taskConfig

The children of the element provide the information about the validation task configuration (i.e. veraPDF Command Line Interface arguments). These settings are reusable across executions and installations. The names of the elements:

- **pdfaFlavor**
- **fixMetadata**
- **collectDetails**
- **reportProgress**
- **stopAfterErrors**

Example

```
<taskConfig>
      <pdfaFlavor>1a</pdfaFlavor>
      <fixMetadata>true</fixMetadata>
      <collectDetails verbosity="5">true</collectDetails>
      <reportProgress>stdout</reportProgress>
      <stopAfterErrors>10</stopAfterErrors>
</taskConfig>
```

TS 4.2.2.3 executionConfig

The children of the element provide the information about the configuration settings unique to a particular validation task. The names of the elements:

- **documentOrigin**
- **output**
- **report**

Example

```
<executionConfig>
      <documentOrigin>https://verapdf.org/Test.pdf</documentOrigin>
      <output>C:/TestFiles/TestFixed.pdf</output>
      <report>C:/TestFiles/TestReport.xml</report>
</executionConfig>
```

TS 4.2.2.4 processMetrics

This element provides information about the time taken to complete a particular task:

- **processStart**
- **processEnd**

Example

```
<processMetrics>
      <processStart>2015-02-28T20:16:12+00:00</processStart>
      <processEnd>2015-02-28T20:16:14+00:00</processEnd>
</processMetrics>
```

TS 4.2.3 validationInfo

The children of the element are described in the table below

Element name	Presence	Description
profile	once	Specifies the Profile that was used for PDF/A Validation
result	once	Results of the performed PDF/A Validation

The **profile** element contains the following children elements:

- **name**
- **hash**

They specify the name and the SHA-1 hash code of the Validation Profile.

Example

```
<validationInfo>
      <profile>
            <name>PDF/A-1a validation profile</name>
            <hash>sha-1 hash code of the profile</hash>
      </profile>
      <result>...</result>
</validationInfo>
```

TS 4.2.3.1 result

The children of the element are described in the table below:

Element name	Presence	Description
compliant	once	The final resolution if the PDF Document is compliant with the given PDF/A Flavor: true or false
statement	once	Textual statement indicating validation result; for example: "The PDF document is not compliant with PDF/A-1a"
summary	once	Brief summary of all the performed checks and fixes. Attributes: ● **passedRules** ● **failedRules** ● **passedChecks** ● **failedChecks** ● **completedMetadataFixes** ● **failedMetadataFixes** ● **warnings**
details	once	Details about the performed Checks of the Validation Rules from the Validation Profile and the related Metadata Fixes

Example

```
<result>
      <compliant>false</compliant>
      <statement>STR_ID_04</statement>
      <summary passedRules="215" failedRules="2" passedChecks="3097" failedChecks="2"
      completedMetadataFixes="1" failedMetadataFixes="1"  warnings="5"/>
      <details>...</details>
</result>
```

TS 4.2.3.1.1 details

The children of the element are described in the table below

Element name	Presence	Description
rules	once	Lists all applied Rules with their statuses
warnings	none or once	Other warnings not related a specific Rule Each warning message is placed in a separate child **warning** element

Example

```
<details>
      <rules>
            <rule id="rule1" status="passed" checks="1">...</rule>
            …
            <rule id="rule53" status="failed" checks="4">...</rule>
            …
            <rule id="rule217" status="passed" checks="2">...</rule>
      </rules>
      <warnings>
            <warning>STR_ID_115</warning>
            <warning>STR_ID_179</warning>
      </warnings>
</details>
```

Depending on its type a Rule may be applied several times in different places of the Document (for example, the Rule like "A stream object dictionary does not contain the F key") so the Checks of a particular Rule are reported. Each Check indicates its status (passed/failed) and location. The status of the Rule is derived from the statuses of its Checks: all must pass in order to have ‘passed’ state for the Rule.

The **rule** element has nested **location** element that specifies the Check location.

The location level is specified by **level** attribute. Possible levels: *document*, *page*; in case of *document* level the only possible nested element is **metadataPath**.

The location of a Check is provided in two forms:

- location in terms of the Validation Model: number of the related indirect PDF object and the validation context
- location in terms of PDF structure and visualization: page id in the report, related resource id in the PDF Features Report, bounding box, XMP location path

Example

```
<rule id="rule1" status="passed" checks="1">
      <check status="passed">
            <location level="page">
                  <context object="11">context</context>
                  <page id="page1"/>
                  <resource>
                        <font id="f1"/>
                  </resource>
                  <bbox llx="100" lly="50" urx="200" ury="70"/>
            </location>
      </check>
</rule>
```

If the Check is somehow related to XMP metadata of the Document or resource (as indicated by 'level' attribute and 'resource' element) then **metadataPath** element may be present and provide the XMP location path to the relevant metadata field in corresponding XMP packet.

A Check may issue a warning message that does not indicate a violation of PDF/A specification, but rather a notification.

Example

```
<check status="passed">
      <location level="document"/>
      <warning>STR_ID_18</warning>
</check>
```

In case of a failed Check there is the error message that explains the reason why the Check failed.

Example

```
<check status="failed">
      <error>STR_ID_305</error>
      <location level="page">
            <context object="11">context</context>
            <page id="page2"/>
            <resource>
                  <font id="f2"/>
            </resource>
            <bbox llx="241" lly="90" urx="321" ury="180"/>
      </location>
</check>
```

A Check may trigger an automatic Metadata Fix attempt. The applicable Metadata Fixes are a part of the Validation Model implementation. The description of the Metadata Fixes can be found in the Validation Profile. If the Metadata Fix attempt is successful then there is the message that provides the details about the performed Metadata Fix. In case of failed fix the message specifies the reason of the Metadata Fix fail.

Example

```
<check status="passed">
      <location level="document">
            <metadataPath>pdf:Keywords</metadataPath>
      </location>
      <fix status="completed">STR_ID_201</fix>
</check>
<check status="failed">
      <error>STR_ID_126</error>
      <location level="document">
            <metadataPath>pdf:Producer</metadataPath>
      </location>
      <fix status="failed">STR_ID_309</fix>
</check>
```

TS 4.2.4 pdfFeatures

The children of the element are described in the table below

Element name	Presence	Description
informationDict	once	Document information dictionary content
metadata	none or once	Document metadata stream content (document-level XMP packet)
documentSecurity	once	The details about Document security

Element name	Presence	Description
lowLevelInfo	once	The low level information about the Document, like number of indirect objects, used filters, Document ID etc
embeddedFiles	none or once	Information about Document embedded files
iccProfiles	none or once	Information about Document ICC profiles
outputIntents	none or once	Information about Document output intents
outlines	none or once	Information about bookmarks
annotations	none or once	Flat list of all the annotations in the Document
pages	none or once	Information about Document pages including all the page resources
documentResources	none or once	Flat list of all the Document resources: graphic states, color spaces, images, XObjects (images and forms), patterns, shadings, fonts, procedure sets and properties dictionaries

TS 4.2.4.1 informationDict

This element contain the list of **entry** elements; each element represents one single key-value pair from PDF Document information dictionary. The dictionary key name is saved as the value of the **key** argument; the dictionary value is saved as the value of the **entry** element.

Example

```
<informationDict>
      <entry key="Title">The document title</entry>
      <entry key="Author">The document author</entry>
      <entry key="Subject">The document subject</entry>
      <entry key="Keywords">keyword1, keyword2</entry>
      <entry key="Creator">The document creator</entry>
      <entry key="Producer">The document producer</entry>
      <entry key="CreationDate">2014-11-23T15:41:28.018+03:00</entry>
      <entry key="ModDate">2014-11-30T21:08:11.397+03:00</entry>
      <entry key="CustomKey">CustomValue</entry>
</informationDict>
```

> NOTE: many of these records may represent the same information as in **documentInfo** element described above. However the data for **documentInfo** element can be taken from various sources, for example from Document XMP metadata stream, in case the XMP modification date is more recent than the Document modification date. On the other hand, **informationDict** element represents the information exactly as it is in the Document information dictionary.

TS 4.2.4.2 metadata

This element contains the document-level XMP metadata package exactly as it is in the original PDF Document or, if automatic XMP metadata fixing is enabled, in the resulting PDF Document. Since XMP serialization is based on XML there is no need to change in the serialized XMP packet, except for encoding. If the encoding used by XMP differs from encoding used for Report generation, the XMP will be re-encoded to make it consistent with the rest of the Report.

Example

```
<metadata>
<x:xmpmeta x:xmptk="Adobe XMP Core 5.2">
      <rdf:RDF>
            <rdf:Description rdf:about="">
                  <xmp:CreatorTool>The document creator</xmp:CreatorTool>
            </rdf:Description>
            <rdf:Description rdf:about="">
                  <dc:format>application/pdf</dc:format>
                  <dc:title>
                        <rdf:Alt>
                              <rdf:li xml:lang="x-default">The document title</rdf:li>
                        </rdf:Alt>
                  </dc:title>
                  <dc:creator>
                        <rdf:Seq>
                              <rdf:li>The document author</rdf:li>
                        </rdf:Seq>
                  </dc:creator>
            </rdf:Description>
      ...
      </rdf:RDF>
</x:xmpmeta>
</metadata>
```

TS 4.2.4.3 documentSecurity

The children of the element are described in the table below.

Element name	Presence	Description
encrypted	once	The scope of the encryption. Possible values: ● *None* ● *All* ● *AllExceptMetadata* ● *OnlyFileAttachments*
method	once	The encryption method. Possible values: ● *No* ● *Password* ● *Certificate*
openPassword	once	The boolean value indicating if there is the password for Document opening
permissionsPassword	once	The boolean value indicating if there is the password for changing Document permissions
printingAllowed	once	Indicates if document printing is allowed. Possible values: ● *No* ● *LowResolution* ● *HighResolution*
changesAllowed	once	The boolean value indicating if it is allowed to modify the Document

Element name	Presence	Description
commentingAllowed	once	The boolean value indicating if it is allowed to comment the Document
fillingSigningAllowed	once	The boolean value indicating if it is allowed to fill in form fields and sign existing signature fields in the Document
documentAssemblyAllowed	once	The boolean value indicating if it is allowed to insert pages into the Document
contentCopyingAllowed	once	The boolean value indicating if it is allowed to copy the content of the Document
contentAccessibilityEnabled	once	The boolean value indicating if it content accessibility feature is enabled
pageExtractionAllowed	once	The boolean value indicating if it is allowed to extract pages from the Document
level	none or once	The encryption level. Possible values: ● *40-bit RC4* ● *128-bit RC4* ● *128-bit AES* ● *256-bit AES*

Example

```
<documentSecurity>
      <encrypted>All</encrypted>
      <method>Password</method>
      <openPassword>false</openPassword>
      <permissionsPassword>true</permissionsPassword>
      <printingAllowed>HighResolution</printingAllowed>
      <changesAllowed>false</changesAllowed>
      <commentingAllowed>false</commentingAllowed>
      <fillingSigningAllowed>true</fillingSigningAllowed>
      <documentAssemblyAllowed>false</documentAssemblyAllowed>
      <contentCopyingAllowed>true</contentCopyingAllowed>
      <contentAccessibilityEnabled>true</contentAccessibilityEnabled>
      <pageExtractionAllowed>false</pageExtractionAllowed>
      <level>128-bit RC4</level>
</documentSecurity>
```

TS 4.2.4.4 lowLevelInfo

The children of the element are described in the table below

Element name	Presence	Description
indirectObjectsNumber	once	The total number of indirect objects in the Document
documentId	none or once	The Document ID that consists of two strings The ID strings are saved as the values of the attributes **creationId** and **modificationId**
filters	none or once	The list of all filters used in the Document The name of the used filter is the value of the attribute **name** of the element **filter**. Possible filer names: ● *ASCIIHexDecode* ● *ASCII85Decode* ● *LZWDecode* ● *FlateDecode* ● *RunLengthDecode* ● *CCITTFaxDecode* ● *JBIG2Decode* ● *DCTDecode* ● *JPXDecode* ● *Crypt*

Example

```
<lowLevelInfo>
      <indirectObjectsNumber>211</indirectObjectsNumber>
      <documentId creationId="B6FB54F3F8554D478DC874F11DAD0F11"
      modificationId="C91F037F8099F24DBB3FF4532DCBEDC8"/>
      <filters>
            <filter name="ASCIIHexDecode"/>
            <filter name="LZWDecode"/>
      </filters>
</lowLevelInfo>
```

TS 4.2.4.5 Embedded files

This element contains the list of **embeddedFile** elements; each of them represents the file that is embedded into the PDF Document.

The children of the **embeddedFile** element are described in the table below

Element name	Presence	Description
fileName	once	The name of the embedded file
description	none or once	The description of the embedded file, if available
subtype	none or once	The MIME subtype of the embedded file, if available

Element name	Presence	Description
filter	once	The filter that is used to encode the file
creationDate	none or once	The embedded file creation date, if available
modDate	none or once	The embedded file modification date, if available
checkSum	none or once	The checksum of the embedded file, if available
size	once	The size of the embedded file

The element has the attribute **id** that contains the ID generated by the PDF parser for this Report entry. This ID uniquely identifies the described object in the given Report and can be referenced from any other part of the Report.

Example

```
<embeddedFiles>
      <embeddedFile id="file1">
            <fileName>data.pdf</fileName>
            <description>This file contains the additional data</description>
            <subtype>application/pdf</subtype>
            <filter>FlateDecode</filter>
            <creationDate>2013-10-21T15:18:32.241</creationDate>
            <modDate>2013-12-15T14:08:17.759</modDate>
            <checkSum>01234567890123456789012345678901</checkSum>
            <size>1234</size>
      </embeddedFile>
      ...
</embeddedFiles>
```

NOTE: this element as well as other elements in PDF Features Report contains only the information provided by PDF parser. A typical use case for this information in Policy Checking is to verify there are no attachments, or there are only attachments of a certain MIME type.There can be more complex Policy requirements like checking if the attached images are valid or attached XML files comply with some Schema. Checking such a requirement involves third-party parsers and validators. According to the Validation Model design these kinds of checks are supported on the level of the Validation Model implementation. They are specified in Validation Profile and performed together with other Validation Checks. The results of these extra checks are included into the Validation Report. Additional Policy Checking can be done based on results of these checks.

TS 4.2.4.6 iccProfiles

This element contains the list of **iccProfile** elements; each of them represents the ICC profile in the PDF Document.

The children of the **iccProfile** element are described in the table below:

Element name	Presence	Description
version	none or once	The profile version, if available
cmmType	none or once	The CMM type, if available
dataColorSpace	none or once	The data color space, if available
creator	none or once	The profile creator, if available
creationDate	none or once	The profile creation date, if available
defaultRenderingIntent	none or once	The default rendering intent, if available
copyright	none or once	The profile copyright, if available
description	none or once	The profile description, if available
profileId	none or once	The profile ID, if available
deviceModel	none or once	The device model, if available
deviceManufacturer	none or once	The device manufacturer, if available

The element has the attribute **id** that contains the ID generated by PDF parser for this Report entry. This ID uniquely identifies the described object in the given Report and can be referenced from any other part of the Report.

Example

```
<iccProfiles>
      <iccProfile id="icc1">
            <version>2.1.0</version>
            <cmmType>type</cmmType>
            <dataColorSpace>RGB</dataColorSpace>
            <creator>The creator</creator>
            <creationDate>1998-02-09T06:49:00.000</creationDate>
            <defaultRenderingIntent>Perceptual</defaultRenderingIntent>
            <copyright>The copyright</copyright>
            <description>The description</description>
            <profileId>1DF3DFD53876AB129CBA7D4A2</profileId>
            <deviceModel>The model</deviceModel>
            <deviceManufacturer>The manufacturer</deviceManufacturer>
      </iccProfile>
      ...
</iccProfiles>
```

TS 4.2.4.7 outputIntents

This element provides the information about the Document output intents

TS 4.2.4.8 outlines

This element provides the information about the bookmarks in the Document

TS 4.2.4.9 annotations

This element contains the list of **annotation** elements; each of them represents the annotation in the PDF Document.

The **annotation** element contains the detailed information about annotation like type, location, references to the annotation resources and other annotations used used by this annotation.

The element has the attribute **id** that contains the ID generated by the PDF parser for this Report entry. This ID uniquely identifies the described object in the given Report and can be referenced from any other part of the Report.

TS 4.2.4.10 pages

This element contains the list of **page** elements; each of them represents the page in the PDF Document.

The children of the **page** element are described in the table below.

Element name	Presence	Description
mediaBox	once	The media box of the page The element uses attributes **llx** (lower left x coordinate), **lly** (lower left y coordinate), **urx** (upper right x coordinate), **ury** (upper right y coordinate) to provide the media box details
cropBox	once	The crop box of the page
trimBox	once	The trim box of the page
bleedBox	once	The bleed box of the page
artBox	once	The art box of the page
rotation	once	The rotation of the page
scaling	once	The scaling of the page
thumbnail	once	The boolean value indicating if thumbnail is present
resources	none or once	Flat list of all the page resources (scanned recursively)
annotations	none or once	Flat list of all the annotations present on the page

The element has the attribute **id** that contains the ID generated by the PDF parser for this Report entry. This ID uniquely identifies the described object in the given Report and can be referenced from any other part of the Report.

Also the **page** element contains the attribute **orderNumber** that provides the order number of the page in the Document.

Example

```
<pages>
      <page id="page1" orderNumber="1">
            <mediaBox llx="0" lly="0" urx="600" ury="800"/>
            <cropBox llx="0" lly="0" urx="600" ury="800"/>
            <trimBox llx="0" lly="0" urx="600" ury="800"/>
            <bleedBox llx="0" lly="0" urx="600" ury="800"/>
```

```
            <artBox llx="0" lly="0" urx="600" ury="800"/>
            <rotation>0</rotation>
            <scaling>1</scaling>
            <thumbnail>false</thumbnail>
            <resources>...</resources>
            <annotations>...</annotations>
      </page>
      ...
</pages>
```

TS 4.2.4.10.1 resources

This element contains the list of the references (by **id** attribute) to the descriptions of all the resources (scanned recursively) used by given page. The list of the resources is flat; it means a resource in this list may be used by some other resource in the list, but this is not anyhow reflected in this part of the Report. The parent-child relations between the resources are provided in resources description in the **documentResources** element of the Report.

The xml structure of the **resources** element is similar to the layout of the resources description in the **documentResources** element. The main difference is that the content of each element representing a certain resource is empty, and there is only **id** attribute present that means this is the reference.

Example

```
<resources>
      <graphicsStates>
            <graphicsState id="gs1"/>
      </graphicsStates>
      <colorSpaces>
            <colorSpace id="cs1"/>
            <colorSpace id="cs2"/>
            <colorSpace id="cs3"/>
      </colorSpaces>
      <fonts>
            <font id="f1"/>
            <font id="f2"/>
      </fonts>
      <images>
            <image id="im1"/>
      </images>
</resources>
```

TS 4.2.4.10.2 annotations

This element contains the list of the references (by **id** attribute) to the descriptions of all the annotations present on given page.

Example

```
<annotations>
      <annotation id="annot1"/>
      <annotation id="annot2"/>
</annotations>
```

TS 4.2.4.11 documentResources

This element describes all the Document resources, separated by resource types.

The children of the **page** element are described in the table below.

Element name	Presence	Description
graphicsStates	none or once	List of all graphics states used in the Document
colorSpaces	none or once	List of all color spaces used in the Document
patterns	none or once	List of all patterns used in the Document
shadings	none or once	List of all shadings used in the Document
xobjects	none or once	List of all XObjects (images and forms) used in the Document
fonts	none or once	List of all fonts used in the Document
procSets	none or once	List of all procedure sets used in the Document
propertiesDicts	none or once	List of all properties dictionaries used in the Document

Each of the elements in the table above contains the descriptions of the resources of the corresponding type. The name of the element representing a resource is the same as the resource type.

Example

```
<documentResources>
      <graphicsStates>
            <graphicsState id="gs1">...</graphicsState>
      </graphicsStates>
      <colorSpaces>
            <colorSpace id="cs1" family="DeviceRGB">...</colorSpace>
            <colorSpace id="cs2" family="Indexed">...</colorSpace>
      </colorSpaces>
      …
</documentResources>
```

The description of each resource contains the references to all the used resources (children) in the **resources** element and references to resources and/or pages which use this resource (parents) in the **parents** element. Unlike the **resources** element in the **page** element the references to children (and parents) only list immediate children (and parents). It is easy to traverse the resources tree up and down using this referencing schema.

If the immediate parent of the resource is **page** it means the resource is used in page content stream.

Example

```
<graphicsState id="gs1">
      <parents>
            <page id="page1"/>
            <page id="page2"/>
            <form id="form1"/>
      </parents>
      …
      <resources>
            <fonts>
                  <font id="f2"/>
            </fonts>
```

```
      </resources>
</graphicsState>
```

The resource element has the attribute **id** that contains the ID generated by the PDF parser for this Report entry. This ID uniquely identifies the described resource in the given Report and can be referenced from any other part of the Report.

In case a resource has associated XMP metadata then the resource element will also include **metadata** element containing the XMP metadata package.

TS 4.2.4.11.1 graphicsState

This element represents ‘graphics state’ resource. The graphics state details include settings for transparency, overprints, fonts etc.

The children elements of the **graphicsState** element:

- **transparency**
- **strokeAdjustment**
- **overprintForStroke**
- **overprintForFill**

Example

```
<graphicsState id="gs1">
      <parents>
            <form id="form1"/>
      </parents>
      <transparency>false</transparency>
      <strokeAdjustment>true</strokeAdjustment>
      <overprintForStroke>true</overprintForStroke>
      <overprintForFill>false</overprintForFill>
      <resources>
            <fonts>
                  <font id="f2"/>
            </fonts>
      </resources>
</graphicsState>
```

TS 4.2.4.11.2 colorSpace

This element represents ‘color space’ resource. The description of each color space shall contain the details relevant for given color space family. The family is specified in **family** attribute. Possible color space families:

- *DeviceGray*
- *DeviceRGB*
- *DeviceCMYK*
- *CalGray*
- *CalRGB*
- *Lab*
- *ICCBased*
- *Indexed*
- *Pattern*
- *Separation*
- *DeviceN*

The children of the **colorSpace** element depend on the color space family.

The children elements in case of *Indexed* color space:

- **base** (reference to the resource representing base color space)
- **hival**
- **lookup**

Example

```
<colorSpace id="cs2" family="Indexed">
      <parents>...</parents>
      <base id="cs1"/>
      <hival>255</hival>
      <lookup>000000 FF0000 00FF00...</lookup>
</colorSpace>
```

The children elements in case of *Separation* color space:

- **alternate** (reference to the resource representing alternate color space)
- **colorName**
- **valuesDefinition** (description of the color space values definition)

Example

```
<colorSpace id="cs3" family="Separation">
      <parents>...</parents>
      <alternate id="cs1"/>
      <colorName>Link blue</colorName>
      <valuesDefinition>...</valuesDefinition>
</colorSpace>
```

The children elements in case of *ICCBased* color space:

- **alternate** (reference to the resource representing alternate color space)
- **components**
- **iccProfile** (reference to an ICC profile from **iccProfiles** element)

Example

```
<colorSpace id="cs4" family="ICCBased">
      <parents>...</parents>
      <alternate id="cs1"/>
      <components>3</components>
      <iccProfile id="icc1"/>
</colorSpace>
```

TS 4.2.4.11.3 xobjects

This element contains the descriptions of 'XObject' resources. There are two types of XObjects: Image and Form.

The children of the **xobjects** element are described in the table below

Element name	Presence	Description
images	none or once	List of all images used in the Document

Element name	Presence	Description
forms	none or once	List of all forms used in the Document

The children elements of the **image** element:

- **width**
- **height**
- **bitsPerComponent**
- **imageMask**
- **maskedImage**
- **filters**

Example

```
<image id="im1">
      <parents>...</parents>
      <width>256</width>
      <height>256</height>
      <bitsPerComponent>8</bitsPerComponent>
      <imageMask>false</imageMask>
      <maskedImage>false</maskedImage>
      <filters>
            <filter>ASCIIHexDecode</filter>
            <filter>JBIG2Decode</filter>
      </filters>
      <resources>...</resources>
      <metadata>...</metadata>
</image>
```

The children elements of the **form** element:

- **bbox**
- **matrix**

Example

```
<form id="form1">
      <parents>...</parents>
      <bbox llx="121" lly="24" urx="168" ury="55"/>
      <matrix>1 0 0 1 0 0</matrix>
      <resources>...</resources>
</form>
```

TS 4.2.4.11.3 font

This element represents ‘font’ resource. The description of each font contains the details relevant for given font type.

The children elements of the **font** element:

- **subtype**
- **name**
- **baseName**
- **firstChar**

- **lastChar**
- **widths**
- **encoding**
- **embedded**
- **subset**
- **fontDescriptor** (the font descriptor describing the font's metrics other then its glyph widths)

Example

```
<font id="f1">
      <parents>...</parents>
      <subtype>Type1</subtype>
      <name>Helvetica-Bold-Font</name>
      <baseName>Helvetica-Bold</baseName>
      <firstChar>0</firstChar>
      <lastChar>255</lastChar>
      <widths>255 255 ... 380</widths>
      <encoding>StandardEncoding</encoding>
      <embedded>false</embedded>
      <subset>false</subset>
      <fontDescriptor>...</fontDescriptor>
</font>
```

TS 4.3 Report example

The self-documented example of the Report prototype: ReportExample.xml

The complete Report schema will be created after the report structure is agreed among all the stakeholders.

TS 5 Policy Profile

The main intention of the Policy Profile is to define Checks of a certain condition imposed on the XML Machine-readable Reports so it is logical to use Schematron language as the basis to define the Profile.

TS 5.1 Schematron overview

Schematron is a rule-based validation language for making assertions about the presence or absence of patterns in XML documents. It is a structural schema language expressed in XML using a small number of elements and XPath.

XPath expressions form the core of the language. They are used in order to formulate rules to check the coherence of XML data.

In a typical implementation, the Schematron schema XML is processed into normal XSLT code for deployment anywhere that XSLT can be used.

Schematron has been standardized to become part of ISO/IEC 19757 - Document Schema Definition Languages (DSDL) - Part 3: Rule-based validation - Schematron.

This standard is available free on the ISO Publicly Available Specifications list:
http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

Schemas that use ISO/IEC FDIS 19757-3 should use the following namespace:

http://purl.oclc.org/dsdl/schematron

ISO “reference” implementation can be found here:

http://www.schematron.com/implementation.html

> NOTE: ISO officially does not endorse reference implementations, however, this version is maintained by the editor of the ISO Standard and developers seeking better understanding of the ISO Standard can reference and follow this implementation.

The “reference” implementation is available as:

- set of XSLT files for XSLT 1.0 and XSLT 2.0
- jar representing Schematron for Apache ANT (new beta)

The veraPDF project intention is to use ISO “reference” XSLT-based implementation that will be driven by an open-source XSLT engine.

The ISO “reference” XSLT-based implementation (called Schematron skeleton) generates the result of validation expressed in Schematron Validation Report Language (SVRL). SVRL is a simple report language defined as part of ISO Schematron. It provides a fairly full set of information from validating a document, and can be used as the basis of subsequent transformations. It is used as the basis for the Machine-readable Policy Checks Report generation.

The validation process consists of the two phases:

1. applying the Schematron skeleton XSLT to Schematron schema to get a new XSLT stylesheet that represents the Schematron schema in XSLT
2. applying the resulting XSLT to the XML document to validate it

Running a Schematron validation in steps using ISO “reference” implementation consists of the following steps:

1. perform XSLT transformation with the arguments:
   input: profile.sch

   output: profile1.sch

   stylesheet: iso_dsdl_include.xsl
2. perform XSLT transformation with the arguments:
   input: profile1.sch

   output: profile2.sch

   stylesheet: iso_abstract_expand.xsl
3. perform XSLT transformation with the arguments:
   input: profile2.sch

   output: profile.xsl

   stylesheet: iso_svrl.xsl
4. perform XSLT transformation with the arguments:
   input: report.xml

   output: policy_checks_result.svrl

   stylesheet: profile.xsl

Any tool that is able to perform XSLT transformations can be used in the steps from above. The phase 1 includes the steps 1, 2 and 3. The phase 2 is represented by the step 4.

Example

Assuming the tool is ‘xsltproc’ the example shell script to perform the validation looks like the following:

```
#!/bin/bash
echo Step 1 ...
xsltproc iso_dsdl_include.xsl profile.sch > profile1.xsl
echo Step 2 ...
xsltproc iso_abstract_expand.xsl profile1.xsl > profile2.xsl
echo Step 3 ...
xsltproc iso_svrl_for_xslt1.xsl profile2.xsl > profile.xsl
echo Step 4 ...
xsltproc profile.xsl report.xml | tee policy_checks_result.svrl
```

More info: http://broadcast.oreilly.com/2009/02/running-schematron-batshell-an.html

> NOTE: there is Schematron Text validator that is based on Schematron skeleton. This validator gives simple text output when errors (failed assertion or successful report) is found. The result is written to output as simple text. More info can be found here: http://www.schematron.com/validators.html

Available Java implementations include:

- http://phax.github.io/ph-schematron/
- https://code.google.com/p/probatron4j/

TS 5.2 Using Schematron for Policy Checks

The Schematron language is rather simple to use. A typical Schematron schema consists of a set of patterns each defining a set of rules that has to be applied to XML document.

> NOTE: the Policy Profile is expected to be written by a user of veraPDF software so we need to be sure the Profile format is easy enough. The Schematron language fits this requirement.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
	<ns uri="http://www.verapdf.org/MachineReadableReport" prefix="vmrr"/>
	<pattern>
		<rule context="...">...</rule>
	</pattern>
	<pattern>
		<rule context="...">...</rule>
	</pattern>
	...
</schema>
```

The rules express specific requirements to the content of the XML file. The requirements are based on XPath.

> NOTE: all the examples below are created based on the ReportExample.xml from the chapter about the Machine-readable Report format.

TS 5.2.1 Policy requirement examples

Policy requirement: the PDF version must be 1.6 or greater.

```
<rule context="/vmrr:report/vmrr:documentInfo/vmrr:pdfVersion">
      <report test="number(current()) &lt; 1.6">
            Policy check error: the PDF version is <value-of select="current()"/>. The
            PDF version must be 1.6 or greater!
      </report>
</rule>
```

For the ReportExample.xml this rule generates the error:

> *"Policy check error: the PDF version is 1.5. The PDF version must be 1.6 or greater!"*

Policy requirement: the validation must be completed without any warnings.

```
<rule context="/vmrr:report/vmrr:validationInfo/vmrr:result/vmrr:summary/@warnings">
      <report test="number(current()) != 0">
            Policy check error: the document was validated with <value-of
            select="current()"/> warnings. The validation must be completed without any
            warnings!
      </report>
</rule>
```

For the ReportExample.xml this rule generates the error:

> *"Policy check error: the document was validated with 5 warnings. The validation must be completed without any warnings!"*

Policy requirement: the Document must be not encrypted.

```
<rule context="/vmrr:report/vmrr:pdfFeatures/vmrr:documentSecurity/vmrr:encrypted">
      <report test="current() != 'None'">
            Policy check error: the document encryption is '<value-of
            select="current()"/>'. The document must be not encrypted!
      </report>
</rule>
```

For the ReportExample.xml this rule generates the error:

> *"Policy check error: the document encryption is 'All'. The document must be not encrypted!"*

Policy requirement: filters JBIG2Decode and JPXDecode are not allowed in the Document.

```
<rule
context="/vmrr:report/vmrr:pdfFeatures/vmrr:lowLevelInfo/vmrr:filters/vmrr:filter">
      <report test="count(current()[@name='JBIG2Decode']) > 0">
            Policy check error: usage of the filter JBIG2Decode in the document is not
            allowed!
      </report>
      <report test="count(current()[@name='JPXDecode']) > 0">
            Policy check error: usage of the filter JPXDecode in the document is not
            allowed!
      </report>
</rule>
```

For the ReportExample.xml this rule generates the errors:

> *"Policy check error: usage of the filter JBIG2Decode in the document is not allowed!"*

> *"Policy check error: usage of the filter JPXDecode in the document is not allowed!"*

Policy requirement: pages in the Document must be neither rotated nor scaled.

```
<rule context="/vmrr:report/vmrr:pdfFeatures/vmrr:pages/vmrr:page">
      <report test="number(current()/vmrr:rotation) != 0">
            Policy check error: the page with id='<value-of select="current()/@id"/>'
            is rotated. The document pages must not be rotated!
      </report>
      <report test="number(current()/vmrr:scaling) != 1">
            Policy check error: the page with id='<value-of select="current()/@id"/>'
            is scaled. The document pages must not be scaled!
      </report>
</rule>
```

For the ReportExample.xml this rule does not generate any errors.

Policy requirement: first page of the Document must not contain images.

```
<rule
context="/vmrr:report/vmrr:pdfFeatures/vmrr:pages/vmrr:page[@orderNumber='1']/vmrr:reso
urces/vmrr:images">
      <report test="count(current()/vmrr:image) > 0">
            Policy check error: the first page contains image with id='<value-of
            select="current()/vmrr:image/@id"/>'. The first page of the document must
            not contain images!
      </report>
</rule>
```

For the ReportExample.xml this rule generates the error:

> *"Policy check error: the first page contains image with id='im1'. The first page of the document must not contain images!"*

Policy requirement: fonts used on the first page of the Document must have 'StandardEncoding' only.

```
<rule
context="/vmrr:report/vmrr:pdfFeatures/vmrr:pages/vmrr:page[@orderNumber='1']/vmrr:reso
urces/vmrr:fonts/vmrr:font">
      <report
      test="/vmrr:report/vmrr:pdfFeatures/vmrr:documentResources/vmrr:fonts/vmrr:font[@
      id=current()/@id]/vmrr:encoding != 'StandardEncoding'">
            Policy check error: the first page uses font with id='<value-of
            select="current()/@id"/>' with encoding '<value-of
            select="/vmrr:report/vmrr:pdfFeatures/vmrr:documentResources/vmrr:fonts/vmr
            r:font[@id=current()/@id]/vmrr:encoding"/>'. The fonts used on the first
            page of the document must have 'StandardEncoding' only!
      </report>
</rule>
```

For the ReportExample.xml this rule generates the error:

> *"Policy check error: the first page uses font with id='f2' with encoding 'WinAnsiEncoding'. The fonts used on the first page of the document must have 'StandardEncoding' only!"*

The complete example of the Policy Profile: PolicyProfileExample.sch

TS 6 Test framework

TS 6.1 Terms and Definitions

Term	Definition
Unit Tests	The set of tests run by developers on compilation and enforced automatically on code check in. Unit tests are generally at class / method level and test a single, simple case. Unit tests are run as part of the developer workflow and shouldn't take so long to run as to hold developers up.
Integration Tests	The set of tests used to test the behaviour of the individual software components and their interaction. Integration tests may take considerably longer to run than unit tests and will typically be run nightly.
Validator Corpora	Test corpora that instantiate a reference interpretation of the PDF/A standards. The corpora consist of files that represent individual requirements of the PFD/A standard.
Policy Checker Corpus	Used to check the function of the policy checking component. These files represent the custom requirements gathered gathered from memory institutions outside of the PDF/A specifications.
Metadata Fixer Corpus	A collection of files that present scenarios correctable by the Metadata Fixer.

TS 6.2 Test corpora

The test file corpora will be curated according to the approach set out in CE 3.2 Corpora and will be developed under revision control in a separate repository to the source code.

TS 6.2.1 Unit test files

A collection of test files created and used by developers and required for running the projects unit tests successfully. These test files must be part of the code base (packaged as Java test resources within the appropriate Maven modules). The files will include examples of:

- PDFs used in unit tests;
- example Policy Profiles used to test the Policy Checker;
- Report Templates used to test the generation of Human-readable Reports.

There may be examples of other file types dependant upon test requirements. This test set will be developed with the software components and will be under revision control alongside the source code.

TS 6.2.2 Validator test corpora

One of the veraPDF consortiums aims is to provide an objective “ground truth” corpus that instantiate the requirements of the various PDF/A specifications. The Validator Corpora will consist of PDF Documents that represent individual requirements derived from the specifications. These corpora will be used to test the veraPDF validator but could also be used to test 3rd party validators.

Testing the validator against the entire set of corpora may take too long to run regularly as unit testing. A nightly QA build on the OPF Jenkins server will compile and test the current master development branch against the validator corpora.

TS 6.2.3 Metadata Fixer test corpus

The fixer test corpus consists of files that are valid PDF/A Documents of some PDF/A Flavour except some type of metadata violation / inconsistency. The Documents represent scenarios where some kind of Metadata Fix or repair resulted in a valid PDF/A.

TS 6.2.4 Policy test corpus

This corpus will represent the Policy Checking requirements gathered from memory institutions. PDF Documents that demonstrate particular Policy issues will be coupled with test Policy Profiles (XML Schematron files) that express a test for the Policy issue.

Testing against the full Policy test corpus isn’t envisaged as a unit testing activity. Depending upon the corpus size and time taken to run the tests this might be a nightly QA build activity.

TS 6.2.5 PREFORMA test corpus

This corpus will be provided by the PREFORMA partners for the final test phase of the project. The functional scope tested by this corpus (Validation, Policy Checking, Metadata Fixer) and any requirements implied by the corpus are currently unknown.

Automated testing of the software against various test corpora is planned to be a business as usual activity. A nightly automated test against the PREFORMA corpus will be set up as soon as the corpus is made available.

TS 6.3 Referenced files

Referenced files are files used to verify test results as opposed to files to perform tests upon. These will consist of various formats, for example:

- PDF Documents that represent cases such as the expected result of a specific Metadata Fix;
- XML files that are the expected Machine-readable Reports for comparison with the those generated as the results of testing;
- HTML and PDF files used to compare to Human-readable Reports produced during testing.

Given that these files often represent the results of testing against particular test files they should be stored with the test data. Reference files for unit testing are packaged as test resources with unit test data files, those for test corpora should accompany the particular the corpora they’re used to verify.

TS 6.4 Automation

TS 6.4.1 Unit testing

Unit tests will be implemented as Junit test cases and suites and run as part of the Maven build (mvn test). This ensures that the test are run as often as the software is compiled. The full set of tests must pass before committing code to master / pushing to GitHub. See CE 3.3 Code for details of the community

contribution guidelines.

TS 6.4.2 Continuous integration

Continuous integration tests, run at code merge with GitHub, will be set up using the Travis continuous service. The first task of this build is to ensure that committed code compiles and if it does to run the unit tests. Travis provides a working Java / Maven environment and so can use the projects standard build and test tools.

During the initial phases of development it will be possible to use Travis to download the various test corpora and run integration level tests on the software. This might become impractical as the corpora grow in size and the time taken to download them and execute the tests increases. The mitigation is to execute these longer running integration tests as part of the nightly QA and release build on the OPF Jenkins server.

TS 6.4.3 Virtualised build/test environment

The project will also include a virtualised continuous integration environment for use by:

- internal developers allowing them to run the full integration test set before pushing code to GitHub;
- external developers wishing to test their contributions to the project; and
- anyone wishing to test the veraPDF software independently.

This virtual test environment will initially be a virtual machine template set up to build and test the users most recent check in using pre-defined corpora and report the results.

TS 7 Internationalization

Internationalization and localization are means of adapting computer software to different languages, regional differences, and technical requirements of a target market.

> **Internationalization** is the process of designing a software application so that it can potentially be adapted to various languages and regions without engineering changes.
>
> **Localization** is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text. Localization (which is potentially performed multiple times, for different locales) uses the infrastructure or flexibility provided by internationalization.

Internationalization of the veraPDF Conformance Checker includes the language-independent design of the internal software logic and interfaces. It means the validation algorithms as well as various kinds of Machine-readable Reports, Validation and Policy Profiles, and Report Templates do not depend on the language that is used to create the final Human-readable Report. Also veraPDF internationalization describes how different languages can be used for Human-readable Reports generation and defines the ways to extend the set of the supported languages.

The library supports different languages and country-specific information on the step of Human-readable Report generation. In more detail:

- the Machine-readable Report is based on English language only;
- the Report Templates convert Machine-readable Reports to a Human-readable Report (see TS 8 Report Template format). The Report Template defines the layout and can be used with different Language Packs;
- the Language Pack specifies all string constants for a given language as well as additional country info (such as date format). The Language Pack has an accessible format, which allows technical translators to create such packs without the need of special programs or tools.

Initial implementation will support a limited number of European languages to demonstrate this mechanism. It is assumed that further translations will be created by the community. Software messaging is controlled by domain experts as described in CE 3.4 Messaging.

The initial release of the veraPDF software will document the localization process in detail, and will allow support for new languages without updating the software.

TS 7.1 Overview

The base technology for veraPDF localization is Translation Memory eXchange (TMX).

TMX is the vendor-neutral open XML standard for the exchange of Translation Memory (TM) data created by Computer Aided Translation (CAT) and localization tools. The purpose of TMX is to allow easier exchange of translation memory data between tools and/or translation vendors with little or no loss of critical data during the process.

The benefits of using TMX:

- it is open standard that is commonly used in many industries;
- it is XML-based so it does not introduce any additional technology which means less complexity for development and maintenance;
- there are existing tools to work with TMX, including free and open-source, so it will be easier to edit and extend the default TMX provided together with veraPDF in order to add new languages.

TS 7.2 Architecture

The veraPDF software is created using English language as the basis. So without the extra localization work the software can generate Human-readable Reports in English.

The software is designed in the way that any text string (that is in English always) that may appear in a Human-readable Report is assigned a string ID. The pairs < string ID, English string > are kept in a separate resource file that is later used for automatic generation of a base TMX file. The string IDs are used instead of the actual text in the Machine-readable Validation Report and in all other similar places. The IDs will be used for searching for an actual localized text string at the moment the Human-readable Report is created.

Examples:

The example entries from the Machine-readable Validation Report

```
<error>STR_ID_305</error>
<warning>STR_ID_70</warning>
<check id="check1">
      <clause>6.1.13</clause>
      <title>STR_ID_501</title>
      <description>STR_ID_502</description>
</check>
```

In this example the error message corresponding to the string ID STR_ID_305 may be a fixed text like “The document catalog dictionary shall not contain a key with the name OCProperties”. Both the ID and the actual text are stored in the resource file and will be automatically placed by the build system in TMX file for future translation.

The messages generated for the Machine-readable Report may be dynamic. It means they are created using some base message with the placeholders which are replaced by some actual values at the moment validation happens.

Validation-related messages are managed by the PDF Validation TWG (see CE 3.4 Messaging).

Example:

```
<error>
      <message>STR_ID_118</message>
      <argument>STR_ID_402</argument>
      <argument>Test title</argument>
      <argument>Unknown title</argument>
</error>
```

In this example the base message defined by STR_ID_118 may be a text like “The value of the %1 metadata entry is not consistent: the document information dictionary contains the value ‘%2’ and the document XMP metadata contains the value ‘%3’”. The placeholders %1, %2, %3,... are to be replaced by the values from argument elements in a corresponding order. The string for STR_ID_402 is “title”. It means the resulting message in the Human-readable Report will look like the following:

> *“The value of the title metadata entry is not consistent: the document information dictionary contains the value ‘Test title’ and the document XMP metadata contains the value ‘Unknown title’”.*

Since the base message and its arguments are separated it is not a problem get the resulting message translated assuming that the translations for the base message and the arguments defined by string IDs are available at the moment the Human-readable Report is generated: first each component is translated and then the placeholders are replaced.

Keeping text strings in a separate resource file and using string IDs everywhere means better control over

the development and the translation processes: we can be sure that any text string intended for a Human-readable Report will eventually get reviewed and translated.

The base TMX file **base.tmx** defines the mapping between string IDs and actual English text strings. Thus the base TMX file does not contain any translations yet, but it can be extended so each English text string is accompanied by a number of its translations to other languages. The base TMX file is actually Language Pack template.

The automatically generated base TMX is passed to a translator that creates the file **default.tmx** containing the pairs < string ID, { English string, German string, French string, ...} >. The resulting TMX file provides the translations for some pre-defined set of languages and is included into the released version of veraPDF. So **default.tmx** file is the Language Pack for all the languages it provides translations for.

> NOTE: It is not a problem to keep them all in a single file as they anyway are installed by default. Alternatively we can split **default.tmx** into separate TMX files so each file contains translations for a specific language only.

When installed veraPDF software has **translations** subfolder in the folder with other veraPDF resources. By default the **translations** subfolder contains two files: **base.tmx** and **default.tmx**. These files are accessible for a user of the software. It means the files can be found easily and the user can modify them in order to support additional languages.

The user can add his own Language Packs into the **translations** subfolder. Although the user can modify original **default.tmx** file the recommended way is to start from the Language Pack template **base.tmx** and then modify it in order to provide translations for a new language. The TMX is open standard and there are a lot of free tools that can be used for editing TMX files, so the user can choose any of them. The resulting new TMX file is the Language Pack that can be shared between different installations of the veraPDF software.

The veraPDF software loads all TMX files available in **translations** subfolder (except for **base.tmx** as it is not expected to contain any translations). When a translated string for a specific string ID is to be found the software first searches in **default.tmx** and then in the rest of the loaded TMX files in the alphabetical order. As soon as the string is found the searching stops and the string is included into the Human-readable Report being created. If the string for given string ID is not found anywhere, the string ID is written in the Human-readable Report as is. This means if for some reason in the Machine-readable Report instead of a string ID the actual text is used, it will move into the Human-readable Report and, although probably not localized, still it is not lost.

The latter may be important for Policy Checking. The Policy Profile syntax is based on Schematron schema syntax that does not provide multi-language support. When defining a rule for Policy Checking a user has to provide the messages which will be reported in the Policy Checks Machine-readable Report in case the rule generates some error. The messages may be static (the text is always the same) or dynamic (the text may change depending on the context of the error).

Examples:

Static message

```
<report test="count(current()[@name='JBIG2Decode']) > 0">
      Policy check error: usage of the filter JBIG2Decode in the document is not
      allowed!
</report>
```

Dynamic message

```
<report test="number(current()) &lt; 1.6">
      Policy check error: the PDF version is <value-of select="current()"/>. The PDF
      version must be 1.6 or greater!
</report>
```

In case of static message instead of the message itself a string ID can be used in the Profile, so this ID will be included into the Machine-readable Report. If the same ID is added into TMX file together with the original message and its translations as described above then it will be correctly resolved during the Human-readable Report generation.

However in case of dynamic message using string ID is not feasible as the actual text message will be different in each particular case. It would be logical to use arguments similar as for validation messages but Schematron schema syntax does not support this.

Another point is that Policy Profiles will be created by the users of the veraPDF software, so for them it may be more difficult to base them on string IDs, then manually map these IDs into actual messages in TMX file. So for them more easy would be to provide actual messages in some chosen language directly in Policy Profile.

The above means that the Policy Checks Machine-readable Report may contain actual messages and on the next stage they simply will be moved into the Human-readable Report which is the correct behavior in given case.

> NOTE: since Schematron is XSLT-based solution it should be possible to localize dynamic messages from Policy Checks Profile using the approach described here: http://www.codeproject.com/Articles/338731/LocalizeXSLT.

TS 7.3 veraPDF TMX format details

The version of the TMX specification that is used by veraPDF is 1.4b: http://www.gala-global.org/oscarStandards/tmx/tmx14b.html

The TMX 1.4b specification is made available by the Localization Industry Standards Association [LISA] under the terms of the Creative Commons Attribution 3.0 Unported (CC BY 3.0) license.

TS 7.3.1 TMX format overview

A TMX document is an XML document whose root element is **tmx**. The **tmx** element contains two children: **header** and **body**. General information about the TMX document is described in the attributes of the **header** element. Additional information is provided in the **note**, **ude**, and **prop** elements. The main content of the TMX document is stored inside the **body** element. It holds a collection of translations contained in translation unit elements (**tu**). Each translation unit contains text in one or more languages in translation unit variant elements (**tuv**). The text of a translation unit variant is enclosed in a **seg** element.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tmx PUBLIC "-//LISA OSCAR:1998//DTD for Translation Memory eXchange//EN"
"tmx14.dtd" >
<tmx version="1.4">
    <header creationtool="TMX Editor"
       creationtoolversion="1.0.1"
       srclang="en-US"
       adminlang="en"
       datatype="plaintext"
       o-tmf="unknown"
```

```
        segtype="sentence">
    </header>
    <body>
       <tu>
             <tuv xml:lang="en-US">
                   <seg>The document catalog dictionary shall not contain a key with
the name OCProperties</seg>
             </tuv>
             <tuv xml:lang="de-DE">
                   <seg>{text in German}</seg>
             </tuv>
             <tuv xml:lang="fr-FR">
                   <seg>{text in French}</seg>
             </tuv>
       </tu>
    </body>
</tmx>
```

The TMX DTD will allow a **tu** to contain a single **tuv** element, and this is used for **base.tmx** because this file does not provide translations yet, and only defines the mapping between string ID and a corresponding text.

Normally a segment helps to translate from one language to another, thus when it comes to default.tmx or a custom TMX file the minimum number of languages in a **tu** element should be two.

TMX format defines two levels:

- Level 1: Only translatable text is included in the TMX document, leaving formatting information aside. This means that **seg** elements do not contain any inline tags.
- Level 2: Text and markup information are included in the TMX document. Inline tags are used to carry formatting information.

veraPDF project uses only Level 1 as all the human-readable text is plain text, so no formatting information is necessary.

TS 7.3.2 Implementation

As described above the first TMX file is **base.tmx**. This file defines the mapping between string ID and actual message in English so this is a Language Pack template. For this purpose the **prop** element is used. The same element is used to specify the type of the text string (“Message”, “Argument”, etc) that is later used at the moment the translation is searched.

Example:

```
<tu>
      <prop type="ID">STR_ID_305</prop>
      <prop type="Type">Message</prop>
      <tuv xml:lang="en">
            <seg>The document catalog dictionary shall not contain a key with the name
            OCProperties</seg>
      </tuv>
</tu>
```

The source language of **base.tmx** is ‘en’. This is the example of base.tmx

Using **base.tmx** as the starting point a translator creates **default.tmx** that contains the translations for any number of languages. The same starting point shall be used by a veraPDF software user in order to add more languages.

If needed the original message that is in general English (‘en’) can be defined more specifically for “en-US”, “en-GB” etc.

Example:

```
<tu>
	<prop type="ID">STR_ID_305</prop>
	<prop type="Type">Message</prop>
	<tuv xml:lang="en-US">
		<seg>The document catalog dictionary shall not contain a key with the name
		OCProperties</seg>
	</tuv>
	<tuv xml:lang="de-DE">
		<seg>{text in German}</seg>
	</tuv>
	<tuv xml:lang="fr-FR">
		<seg>{text in French}</seg>
	</tuv>
</tu>
```

This is the example of default.tmx

As explained above a user can add custom TMX files with more languages.

The identifier like ‘en-US’ describes generic language (‘en’) together with its specific language (‘US’). When a translation is searched at the moment the Human-readable Report is generated, veraPDF tries to find the **tuv** element with the **xml:lang** attribute that exactly matches the language requested by the user for the Report generation. In case there is no entry with the exact match but there is the entry with matching generic language veraPDF may choose to use this entry. In this case the priority is given to the entry with generic language only.

TS 7.3.3 Tools

This sub-clause lists the tools which can be used to work with TMX files.

- TMXValidator: “checks your documents against TMX DTD and also verifies if they follow the requirements described in TMX specifications”
- Heartsome TMX Editor: “This is the powerful TM maintenance tool for all CAT software”

TS 7.3.4 Additional locale information

Localization requires not only translated text messages, but also other information like numbers, dates etc formatted according to the local rules.

The formatting rules shall be the part of the TMX file. For example, in case of numbers the special patterns are added into the **default.tmx** defining the formatting for each specific locale.

Example:

```
<tu tuid="NumberFormat">
      <tuv xml:lang="en">
            <seg>#,###.00</seg>
      </tuv>
      <tuv xml:lang="es">
            <seg>#.###,00</seg>
      </tuv>
      <tuv xml:lang="fr">
            <seg># ###,00</seg>
      </tuv>
</tu>
```

The patterns define the way how space, dot and comma shall be used to format number in each specific case. The similar patterns can be defined for dates formatting.

TS 8 Report Template format

TS 8.1 Overview

A Report Template defines the Human-readable Report layout, artworks, fonts, and other formatting details. It also controls the verbosity level and allows filtering out certain types of messages or grouping them by severity or object type. The information that will be included into the resulting Report is anyway limited by the verbosity requested when generating source Machine-readable Report.

The Report Template itself does not include the exact text messages to be used for Human-readable Report. There is a Language Pack that provides such messages in the expected language, see TS 7 Internationalization and CE 3.4 Messaging.

Since the syntax for Machine-readable Reports is XML the following technologies are the base for the Report Templates:

- XSLT with a third-party XSLT transformation tool - for a Report Template to convert to an HTML Human-readable Report, for example used in FS 4.2.1 Web Graphical User Interface (GUI-W);
- XSL-FO with a third-party FO-processor (for example, Apache FOP) - for a Report Template to convert to PDF Human-readable Report (see Annex E.4.4 Reporter).

The third-party XSLT engine for converting to HTML can be the same as used for applying Schematron validation rules defined by Policy Profile (see TS 5 Policy Profile).

In order to convert XMP metadata into a human-readable form a Template can additionally use XMP location path syntax.

In case the target format is HTML the resulting Report can be either a single HTML file with all the resources embedded (CSS, JS, images) or a folder containing a base HTML file and all other resources as external files. The Report Template for HTML generation shall provide an option to control this.

TS 8.2 Accessibility

Human-readable Reports will follow accessibility principles stated in ISO 9241-171:2008.

Human-readable Reports in HTML will comply with WCAG 2.0 Level AA.

See also FS 4 Interfaces for a discussion of the approach to user-centred design.

TS 9 Integration with third-party tools

The ability to integrate with other applications is a key aspect of the Conformance Checker extensibility.

TS 9.1 Overview

The product of parsing the PDF Document and extracting the Embedded Resources (image, font, colour profiles, etc.) is data is packed into an intermediate format understandable by Embedded Resource Parsers which are third-party software tools. These tools process the Embedded Resource and return an Embedded Resource Report as described in FS 3.1 Embedded Resources.

To enable integration with a third-party libraries/application/services it is necessary to:

- define the interaction interface to be used between Conformance Checker and a third-party tool;
- describe the ways to adapt third-party tools interfaces to the defined interaction interface.

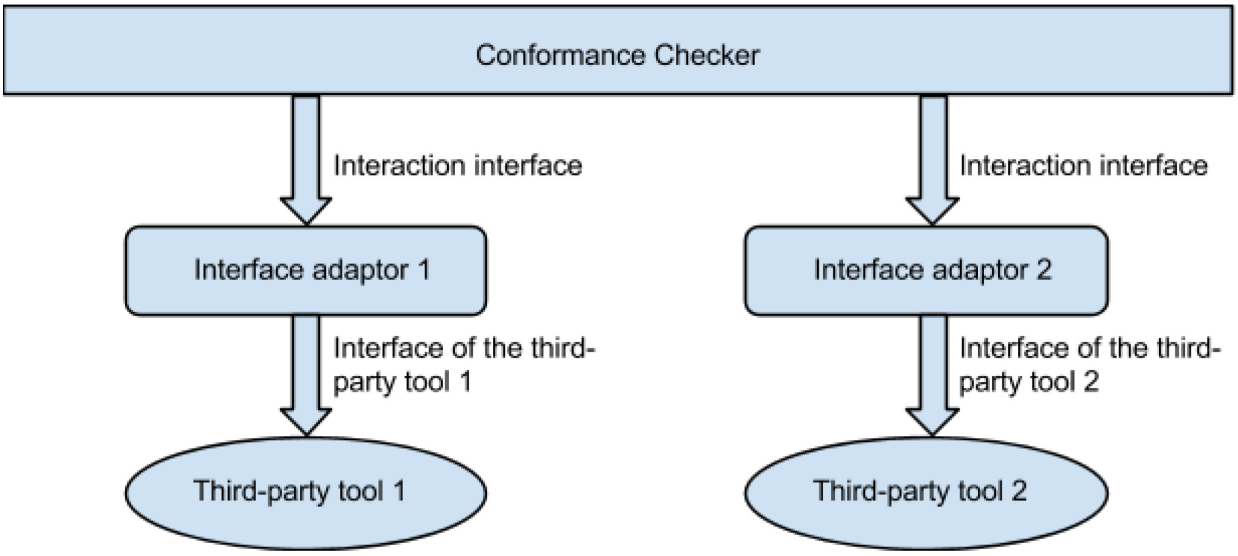


Figure 1: Third-party tools integration

Figure 1 show the interaction schema. The Conformance Checker exposes the **interaction interface** that shall be supported by any third-party in order to be incorporated into validation process. Normally all third-party processing utilities and libraries for processing images, fonts, and other resources publish their own interfaces which are far from what the Conformance Checker requires. Then it is necessary to create **interface adaptors** which adapt the **interface of** each selected **third-party tool** to the interaction interface.

The interaction interface is published by components of the Conformance Checker: the Command Line Interface (CLI) and Library. The third-party tool interfaces may vary depending on the tool design and implementation. The interface adaptor creation is the responsibility of the user who wants to integrate a certain third-party tool into the Conformance Checker validation process.

TS 9.1.1 Command Line Interface

The **interaction interface** exposed by CLI defines that in case of an Image XObject the image data stream is saved into the temporary external binary file. The image metadata as well as the required image check is saved as the configuration XML file. Also the interaction interface declares that the paths to these temp files are passed as the first and second command line arguments for a third-party command line interface utility. The third parameter is the path to not yet existing report file that have to be created by the utility. The CLI starts the utility, waits till it finishes processing and reads the results from the saved report. The interaction interface describes the way how such an utility can be “registered” in CLI: the path to it is specified as one of the arguments of the command to execute CLI validation.

The third-party tool that a CLI user wants to use to check the image is, for example, the dynamic library with some published interface (**third-party tool interface**). Then the required **interface adaptor** is the command line interface utility that accepts the paths to temp files as arguments, loads the third-party dynamic library, performs the requested checks and saves the results in the report file.

Alternatively instead of temporary files the interaction interface may specify stdin/stdout pipes or TCP/IP connections as the communication channels. The third-party tools may be command line interface utilities, dynamic libraries, COM objects, web services, hot folder applications, etc.

TS 9.1.2 API Interface

The **interaction interface** exposed by the Library is a set of Java interfaces (which shall be implemented) as well as environment objects (for example, logger object to report results of the checks). The interfaces use PDF objects like Stream, Resource, etc.

The third-party tool is, for example, a REST (**third-party tool interface**) service accessible over the Internet that is able to perform fonts checks. In given case the required **interface adaptor** is the set of implementations of the Java interfaces. The implementation accepts font object as a Resource, sends the font data in HTTP request composed according to the service REST API documentation and logs all the results using the logger object.

Like in the case of the CLI interaction interface the third-party tools may be command line interface utilities, dynamic libraries, COM objects, web services, hot folder applications, etc.

Annex A: Communications Plan

A.1 Aims and objectives

The Communications Plan is based on a set of channels addressing distinct veraPDF stakeholder groups, (see CE 1 Stakeholders). Table 1 details the means by which the veraPDF Consortium's channels will reach their respective audiences.

Table 1: Channels for veraPDF communications with stakeholders

Channel	Stakeholders	Description	Reach (2015-01)
Websites	All	The veraPDF Consortium will build and maintain a dedicated web presence at http://verapdf.org/ to host all other online sources (mailing lists, blogs, software infrastructure, etc.). The website will be produced by a professional design agency.	*[to be determined]*
	All	An online demonstrator of the veraPDF Conformance Chcker at http://demo.verapdf.org/ (see Annex F)	*[to be determined]*
	Industry	Web content rooted at http://www.pdfa.org/veraPDF informing visitors about industry support for veraPDF	PDFA website averages c.16,000 users/month
	Industry partners	The websites of PDF Association members and Partner Organizations may be used to promote veraPDF	*[to be determined]* Likely to be broad, based on users of partner websites
	Memory institutions	Web content rooted at http://openpreservation.org/about/projects/verapdfa/ to provide information about the project in the context of other OPF activity (interest groups, events, software, etc.)	OPF website averages c.3,000 users/month

Channel	Stakeholders	Description	Reach (2015-01)
Blogs	All	The PREFORMA project has a blog on the Digital Meets Culture website. We will produce content for this blog providing project updates and provide the option to syndicate blogs from the Open Preservation Foundation website.	*[to be determined]* Analytics to be sourced from PREFORMA
	Industry	The PDF Association website hosts several news and other blogs, and actively promotes this content on LinkedIn, Twitter and other social media platforms. In addition, participating vendors will find it in their interest to promote information and awareness of veraPDF in their own communications.	*[to be determined]* Likely to be broad; varies based on degree of industry participation in the project.
	Memory institutions	The Open Preservation Foundation website blogging platform will provide progress updates and discuss issues of interest to memory institutions	Views for blog posts in the range several hundred to a few thousand
Mailing lists	All but customers	Dedicated mailing lists to allow testers and then users to remain current with veraPDF news and developments	*[to be determined]*
	Industry	The mailing list of the PDF Association PDF Validation TWG (as of 2015-02-06).	48 subscribers (from 28 companies)
	Industry	The mailing list of the PDF Association membership (as of 2015-02-06)	426 subscribers (from 106 companies)
	Memory institutions	Open Preservation Foundation mailing lists for members and digital preservation community	707 subscribers (OPF community); 1,483 subscribers (DPC community)
Social media	All	The veraPDF Consortium will use the #veraPDF hashtag on social media, and will drive content with a dedicated Twitter account	*[to be determined]* Analytics will be monitored for use of the hashtag
	Industry	https://twitter.com/PDFAssociation	290 followers
	Memory institutions	https://twitter.com/openpreserve	960 followers

Channel	Stakeholders	Description	Reach (2015-01)
Webinars	Industry, 3rd party standards	The PDF Association held two webinars for PDF Association members on 2014-11-26 and 2014-12-02 to introduce veraPDF and the PDF Validation TWG. Several webinars promoting veraPDF will be held in Phase 2.	Over 60 attendees in total; 24 views of the recording as of 2015-02-06
		PDF Association educational and industry news webinars for members and the public	Periodic, with 10-35 attendees
	Memory institutions	Open Preservation Foundation held a webinar for members in February 2015 to present the functional and technical specifications. Several webinars promoting veraPDF will be held in Phase 2.	8 attendees (from 8 institutions) plus views of the recording (via email, not tracked)
Promo material	All	Brochures, T-shirts, contributor awards and other collateral as deemed necessary.	*[to be determined]* Stakeholders interacting directly with the project
Publications	All	Consortium partners will publish papers and articles at selected conference or relevant academic journals, including a new edition of PDF/A in a Nutshell (free to download) and other distributable publications and slide templates	Over 10,000 printed copies of PDF/A in a Nutshell have been distributed
Individual contacts	Industry Memory institutions 3rd party communities Commercial customers	veraPDF Consortium members, especially PDF Association and Open Preservation Foundation Executive Directors, will engage individual stakeholders and customers, including government agency representatives, with emails, phone calls and in-person meetings	*[to be determined]* Responsive to particular need

A.2 Conferences and events

The Communications Plan includes resources for marketing veraPDF at conferences, workshops and other events (see CE 2 Community development activities). Table 3 details existing and planned events by which the veraPDF Consortium's channels will reach their respective audiences.

PREFORMA Evaluation Criteria

D8.1 *(vi) marketing at conferences: technology providers market the reference implementation and conformance checker at conference for professional networks of developers and digital preservationists;*

Table 2: In-person events in which veraPDF will be actively promoted.

Channel	Audience	Description / Example	Reach
Conferences	Industry	The PDF Association will hold two technical conferences in 2015 with sessions on veraPDF (June 2015 in Cologne, Germany and October, 2015 in San Jose, California). veraPDF information and updates will be a regular fixture at future PDF Association events	Developers (not only PDF Association members) in Europe and North America. Recordings of VeraPDF sessions will be made public and promoted
		The biannual ISO 19005 and ISO 32000 Committees meetings	ISO committee members
		3rd party vendor-oriented events in which the PDF Association has previous or planned participation, including AIIM, CeBIT, LegalTech, Document Strategy Forum, Xplor International and others	Implementers of PDF software in document management, transactional processing and more
		The PDF Association Executive Director presented a paper on PDF/A validation at iPres 2014	Professional digital preservationists
	Memory institutions	Open Preservation Foundation will submit papers to digital preservation conferences, including - iPres 2015 in Chapel Hill, US - Scientific Archivists Group conference in Cardiff, UK - Digital Library conference, Slovakia	International digital library and preservation community
	Commercial customers	User-oriented “PDF Day” and similar events at 3rd party forums. Three such events were conducted in 2014.	Users of PDF and PDF-enabled software
Workshops	Industry	The PDF Association will hold educational workshops in Phase 2 to encourage vendor participation and adoption.	PDF software developers
	Memory institutions	Open Preservation Foundation will hold requirements workshops early in Phase 2 to gather policy requirements. A group of contributors and evaluators will be identified to work with veraPDF on creating Policy Profiles, providing test files, and testing the software.	Larger and smaller memory institutions

Channel	Audience	Description / Example	Reach
Webinars	Industry	PDF Association informational and industry news webinars for members.	106 member companies of the PDF Association
	Memory institutions	Open Preservation Foundation webinars for members and the digital preservation community (free to attend).	Larger and smaller memory institutions
	Commercial customers	During Phase 2 the PDF Association will hold educational webinars and present at vertical industry events in to encourage customer awareness.	PDF software implementers and end users

Annex B. Technical Milestones and Deliverables

B.1 Phase 1 planning

Milestone	Dates	Description	Deliverables
M1	28/02/2015	End of Phase 1	
	16/04/2015	Start of Phase 2, start of 1st prototyping phase	
M2.1	15/07/2015	Internal checkpoint during 1st prototyping phase	Generic veraPDF Validation Model PDF Parser (Prototype) Machine-readable Reports (RC) PDF Features Report (Prototype) Collect Institutional Policy requirements 3rd party Integration (Interface) Shell (Requirements) Test Corpora (PDF/A-1)
M2.2	30/10/2015	End of 1st prototyping phase, start of redesign phase	PDF Parser (RC) Conformance Checker (PDF/A-1b) PDF Features Report (RC) Metadata Fixer (Prototype) Policy Checks (Prototype) Test profiles for Institutional Policy requirements Language Packs support Human-readable Reports (HTML5) 3rd party Integration (Prototype) Shell (Prototype) Test Corpora (Internal Review)

Milestone	Dates	Description	Deliverables
M2.3	28/02/2016	End of redesign phase, start of 2nd prototyping phase	Institutional Policy Requirements (OPF Review) Test Corpora (PDF Association Review)
M2.4	20/12/2016	End of Phase 2 (also end of 2nd prototyping phase)	Conformance Checker (RC) Metadata Fixer (RC) Policy Checks (RC) Human-readable Reports (PDF) Shell (RC) Integration PREFORMA Partners Test Corpora (RC) Institutional Policy Requirements (RC)

B.2 Phase 2 Planning

ID	Story / Task	Depends on	Milestone	Notes
1	***Generic Model implementation***			
1.1	Parsing the model description		M2.1	
1.2	Helper classes to access all information on the model: type hierarchy, properties, associations	1.1	M2.1	
1.3	Parsing the validation profile		M2.1	
1.4	Generic validation algorithm: navigate through all objects starting from the root object and following all association rules	1.2, 1.3	M2.1	
1.5	Evaluation of conditional expressions	1.2	M2.1	
1.6	Message generation	1.2, 1.3	M2.1	
1.7	Global storage (variables) support	1.3	M2.2	
2	***Implementation of PDF Parser***			
2.1	Define the PDF Model in the formal syntax		M2.1	
2.2	*PDFBox based implementation of the PDF Parser*			
2.2.1	Implementation of CosObject types (all properties and associations)		M2.1	
2.2.2	Implementation of PDObject types for PDF/A-1b		M2.1	
2.2.3	Implementation of PDObject types for all PDF/A Flavors	2.2.1, 2.2.2	M2.2	

ID	Story / Task	Depends on	Milestone	Notes
2.2.4	Implementation of Operator types for PDF/A-1b		M2.1	
2.2.5	Implementation of Operator types for all PDF/A Flavors	2.2.1, 2.2.4	M2.2	
2.2.6	Implementation of External types for all PDF/A Flavors		M2.2	
2.3	Greenfield implementation of PDF Parser	2.1	M2.4	
3	***Implement PDF/A Validation Profiles***			
3.1	Prototype COS Level checks	1, 2.1	M2.1	
3.2	Prototype PD Level checks	1, 2.2.1	M2.1	
3.3	Prototype checks for Operator types	1, 2.2.4	M2.1	
3.4	Prototype checks for External types	1, 2.2.6	M2.2	
3.5	Final version of PDF/A-1b profile	3.1-3.4	M2.2	
3.6	Final version of PDF/A-1a profile	3.1-3.5	M2.4	
3.7	Final version of PDF/A-2b profile	3.1-3.5	M2.4	
3.8	Final version of PDF/A-2u profile	3.7	M2.4	
3.9	Final version of PDF/A-2a profile	3.6	M2.4	
3.10	Final version of all PDF/A-3 profiles	3.7-3.9	M2.4	
4	***Generate the Machine-readable Validation Report***			
4.1	Create Validation Report		M2.1	
4.2	Prototype PDF Features Report		M2.1	
4.3	Final version of the PDF Features Report	4.2	M2.2	
5	***Metadata Fixer implementation***			
5.1	Prototype Metadata Fixer		M2.2	
5.2	Final version of Metadata Fixer		M2.4	
6	***Policy Checker implementation***			
6.1	*Support Schematron profiles*			
6.1.1	Choose between XSLT2, XSLT1, or javax.xml.validation implementations		M1	
6.1.2	Select small set of policy examples for implementation		M2.1	
6.1.3	Prototype of Java based Schematron Validation	6.1.1	M2.2	
6.1.4	Test prototype using real institutional policy requirements	6.3	M2.3	
6.1.5	Produce release candidate for Schematron support	6.1.3	M2.4	
6.2	*Generate Machine-readable Policy Report*	6.1		
6.2.1	Define machine readable report format		M2.1	
6.2.2	Generate MR report from Java prototype	6.1.3	M2.2	
6.3	*Create test cases representing institutional policy requirements*			
6.3.1	Gather requirements from OPF / DPC membership		M2.1	

ID	Story / Task	Depends on	Milestone	Notes
6.3.2	Produce test files representing member requirements	6.3.1	M2.2	
7	***Human-readable Report generation***			
7.1	Support TMX-based Language Packs		M2.2	
7.2	*Support Report Templates*			
7.2.1	HMTL5 reports		M2.2	
7.2.2	PDF reports		M2.4	
7.3	Create sample Language Packs		M2.4	
8	***API for third-party plug-ins***			
8.1	Interface for passing data to the plug-ins and receiving error code + report		M2.1	
8.2	*ICC Profile Validation*			
8.2.1	Integration of the ICC profile validator from SmapleICC	8.1	M2.2	
8.2.2	Develop veraPDF ICC Validator	8.1	M2.4	Upon budget availability
8.3	*JP2K Validation*			
8.3.1	Integrate the existing Python validator	8.1	M2.2	
8.3.2	Integrate the Validator from OPF partner	8.1	M2.4	
8.4	*Font Validation*			
8.4.1	Type1 Validator	8.1	M2.4	Upon budget availability
8.4.2	CFF Validator	8.1	M2.4	Upon budget availability
8.4.3	TrueType Validator	8.1	M2.4	Upon budget availability
8.4.4	OpenType Validator	8.1	M2.4	Upon budget availability
9	***Shell implementation***			
9.1	*Interface for passing data to validators and receiving error code + report*			
9.1.1	Draft prototype interface based on API for plug-ins		M2.1	
9.1.2	Clarify PREFORMA consortium requirements for a shell		M2.1	
9.2	*Implement prototype Shell integrations*			
9.2.1	Prototype veraPDF integration	3.5, 9.1	M2.3	
9.2.2	Prototype ICC validation integration	8.2, 9.1	M2.2	
9.2.3	Prototype JP2K validation integration	8.3, 9.1	M2.2	
9.3	Integration with other PREFORMA conformance checkers	9.2	M2.4	Dependent on other suppliers
10	***Test Corpora***			
10.1	Description of all required test cases		M1	
10.2	Enhanced corpora for PDF/A-1		M2.1	
10.3	Test corpus for PDF/A-2		M2.2	
10.4	Test corpus for PDF/A-3	10.3	M2.2	

ID	Story / Task	Depends on	Milestone	Notes
10.5	Test corpus for Tagged PDF		M2.2	
10.6	Test corpus for Policy Checks	6.3	M2.2	
10.7	Test corpus for Metadata Fixer		M2.2	
10.8	PDF Validation TWG approval for PDF/A Validation Corpora	10.1-10.5	M2.3	
10.9	Publish final approved corpora under open license	10.1-10.8	M2.4	

Annex C: PDF/A Test Corpora Analysis

C.1 PDF/A Test Suite

The PDF/A test Suite contains the detailed analysis of all normative PDF/A requirements (all versions and all levels) formalized in terms of test cases required to verify the definitive Implementation Checker.

It also indicates which of these test cases are covered by the existing test corpora:

- Isartor Test Suite
- Bavaria Test Suite
- BFO Test Suite

All test cases come only with the textual description. Actual test files will be created during Phase 2 of the PREFORMA project.

The exact meaning of the columns in the table below are:

A. (№:) Sequential number of the test case
B. (Isartor / Bavaria / BFO) The ID of the test file covering this test case (if any).
C. (PDF/A Specification) Reference to the relevant PDF/A Specification section
D. (Description) Relevant clauses in the specifications
E. (Test case) Test case condition
F. (Version / Level) Version (1,2,3) and Level (a,b,u) of PDF/A specification this test case is applicable
G. (Example) A sample for the test PDF document to be case for this test case
H. (Status of Test Case) Indication of whether this test case is positive (pass) or negative (fail)

In total, there are 719 test cases covering all three versions of PDF/A standards.

C.2 Tagged PDF Test Suite

The Tagged PDF Test Suite contains a similar analysis of all “shall” clauses within two sections of PDF specifications covering the requirements for PDF/A Level A conformance.:

- PDF Version 1.4, Subsection 9.6 “Logical Structure”, Subsection 9.7 “Tagged PDF”
- ISO 32000-1, Subsection 14.7 “Logical Structure”, Subsection 14.8 “Tagged PDF”

Similar to the PDF/A Test Suite it contains the following data (columns of the table):

A. (№:) Sequential number of the test case
B. (ISO 32000-1) Reference to the relevant section of ISO 32000-1 Specification (the same clauses do exist also in PDF 1.4 Specification).
C. (Description) Relevant clauses in the specifications
D. (Policy) Identification of the validation policy (Ignore / Machine /Human). “Ignore” value means that this clause is either generic and does not require any validation or refers to the conforming reader. “Machine” value indicates that this clause may be validated in an algorithmic way without human intervention. “Huma” means that validation of this clause requires human intervention.
E. (Test case) Test case condition
F. (Status of Test Case) Indication of whether this test case is positive (pass) or negative (fail)

In total there are 110 test cases including 69 cases with “Machine” policy.

C.3 PDF/A “Should” and “May” Clauses

Finally, we also analyse all “*should*” and “*may*” clauses in PDF/A specifications and list the PDF Features we will include into the PDF Feature Report so that these clauses can be verified as a part of Policy Checks.

The report on “Should” and “may” clauses contains the following data (columns of the table):

A. (№:) Sequential number of the test case
B. (Section) Reference to the relevant section of PDF/A Specifications
C. (Description) Relevant clauses in the specification
D. (Clause type) Identification of the clause type (May, Should, Conforming reader, Conforming writer)
E. (Policy) Identification of the validation policy (Ignore / Machine /Human). “Ignore” value means that this clause is either generic and does not require any validation or refers to the conforming reader. “Machine” value indicates that this clause may be validated in an algorithmic way without human intervention. “Huma” means that validation of this clause requires human intervention.
F. (PDF Feature) The relevant data of the PDF Features report

In total there are 94 clauses identified within all PDF/A-1, 2, 3 Specifications.

The complete Test Corpus report is presented in the three Excel spreadsheets attached to this Report. Each of the spreadsheets covers one of the above parts of the Test Corpus report.

Annex D: PDFBox Feasibility Study

D.1 Summary and recommendations

The overall quality of the PDFBox libraries considered for use is acceptable but we'll be taking precautions, for example adding unit tests for crucial PDFBox parsing dependencies, using code coverage tools to help isolate untested code.

The XMPBox library is well tested and re-implementing would be a case of re-inventing a very serviceable XMP wheel. Preflight PDF/A compliance checks could be used adding unit tests for key checks we depend upon.

The adoption of Fontbox is uncertain, test coverage is poor and there's a large code base that's quite interdependent. Testing code like this can be harder than rewriting it. Under the circumstances we'll proceed with suspicious caution.

D.2 Legal information

PDFBox is licensed under the Apache License, Version 2.0. Apache provides an analysis of software dependencies[1] which we refer to in Annex E: License Compatibility. Please refer to that document for the full legal analysis of license compatibility.

D.3 Versions and update policy

Current stable version: 1.8.7

PDFBox 1.8.8 release: 09.12.2014

Previous versions: http://archive.apache.org/dist/pdfbox/

Average update rate: ~3 releases per year

Project page	https://pdfbox.apache.org/
API documentation	http://pdfbox.apache.org/docs/1.8.6/javadocs/
Source code	http://svn.apache.org/viewvc/pdfbox/
Mailing list	https://pdfbox.apache.org/mailinglists.html
Issue tracking	https://issues.apache.org/jira/browse/PDFBOX
Static code analysis	https://analysis.apache.org/dashboard/index/org.apache.pdfbox:pdfbox-reactor?did=9

[1] https://analysis.apache.org/plugins/resource/58986?page=org.sonar.plugins.design.ui.libraries.LibrariesPage

D.3.1 Version 2.0

New features:

- Completely rebuilt font parser.
- Pattern rendering
- Pages resource caching
- Font embedding
- Parsing
- Page Tree
- Text extraction on Java 8

All this changes are affecting pdfbox api.

Expected release date: first half of 2015 based on the number of remaining tasks in Jira and the average time it takes to resolve them.

D.4 PDFBox open source project analysis

This section presents an overview and analysis of the PDF Box project, concentrating on the code base. Currently the information comes from two publicly available sources:

- the Open Hub page for PDFBox, Open Hub provide metrics and analysis for the comparison of open source projects.
- Apache's Sonar instance for the PDFBox project which provides static analysis of the PDFBox code base.

For context we make comparisons with two other open source Java projects:

- Apache Tika, a content/metadata extraction application that calls PDFBox amongst others; and
- Apache Maven, a production class Java software project management tool used by developers everywhere. I'd expect this project to follow best practises and have strong metrics as it's a tool written by developers, for developers.

Both projects are established and have Java code bases of significant size. Tika is expected to provide a "lower bound" and is not exemplary with respect to unit test coverage and elegance of design. Maven handles a complex task reliably with a flexible architecture that's encouraged the growth of a huge ecosystem of Maven plugins. The expectation is that Maven will provide the "upper bound" exemplar of a high quality project.

The meaning of the particular measures compared and why they were chosen will be given in context. Some of the measures are inexact and aren't to be taken literally, for example Open Hub's estimations of effort used are always alarmingly high. Others, like unit test coverage, offer a more reliable and objective indicator of software quality.

D.4.2 PDFBox and Git/GitHub

VeraPDF fork of PDFBox here: https://github.com/verapdf/pdfbox, this was forked from the official Apache mirror: https://github.com/apache/pdfbox, the upstream repository. The Apache mirror is a mirror of the official Apache Git repo: git://git.apache.org/pdfbox.git that in turn mirrors the Apache PDFBox SVN repository. Because of it's SVN roots the git repo has no master branch, the conventional main branch for git repos. Instead the git repo follows the SVN convention of using trunk as its main branch.

D.4.2.1 Candidate process for working with PDFBox mirror

A working assumption is that we'll only be submitting bug fixes for PDFBox. A fix should be fine grained and raised as an issue on PDFBox JIRA.

- trunk is reserved for synch from upstream;
- verapdf add master branch reserved our production HEAD, normally only updated via pull requests; and
- all work performed on development branches named after issue addressed.

Developers create a branch per-issue worked on. First implement a failing unit test to illustrate the issue clearly, then the fix. Once finished (tests passing) pull and merge current master to their branch to catch any changes. It's the VeraPDF teams responsibility to sort out any merge conflicts, though we may have to enlist upstream help from time to time.

The above is simply a re-working of the standard GitHub workflow[2] adapted to account for the differences in convention between SVN and Git.

D.4.3 Continuous integration & static code analysis

D.4.3.1 Travis-CI

The project already had a Travis-CI build file, added the VeraPDF instance: https://travis-ci.org/verapdf/pdfbox. Amended the build to exclude the GitHub pages branch, and to add an OpenJDK 7 build to the existing Oracle 7 and OpenJDK 6 builds. These all built straightforwardly using the standard Travis Maven setup.

D.4.3.2 OPF Jenkins

Added PDFBox as a nightly build on the OPF Jenkins server: http://jenkins.opf-labs.org/job/PDFBox/. This uses Oracle JDK 7 to build PDFBox. Initially a few tests failed due to the lack of Java cryptographic extensions. Installing these still left some skipped tests. The travis build suggests installing the fonts-liberalism package for Ubuntu boxes. The OPF Jenkins server is an Ubuntu server and all tests run and pass once the fonts-liberalism package was installed,

D.4.3.3 OPF Sonar

The nightly Jenkins build also triggers the OPF sonar server to perform static analysis. This crucially includes unit test coverage, not available on the Apache instance. http://sonar.opf-labs.org/dashboard/index/6820

D.4.4 Code quality

D.4.4.1 Key metrics

Static code quality analysis is an inexact science. While numeric measures don't give a complete picture there are a few key metrics that can provide a good initial indicator. We've used four metrics for assessing code quality and a simple source line count to provide an indicator or project size. Some of them provide multiple measures, the rest of this section describes the metrics used and the rationale for choosing them.

D.4.4.1.5 Project size

This isn't a measure of quality, but it needs to be considered when comparing quality metrics.

[2] https://guides.github.com/introduction/flow/index.html

There's a variety of measures, the meaning of many are obvious:

- lines of code excluding blank lines:
- number of files;
- number of classes;
- number of functions; and
- development effort in man months.

The last is calculated by an Open HUB FOSS software portal[3] and is not meant to be taken literally.

D.4.4.1.2 Unit test coverage

Unit test coverage, or code coverage calculates the degree that the source code of an application is tested by a test suite. We'll look at three coverage measures and the overall success:

- Condition coverage measures the percentage of logical paths through the code that are executed by the tests.
- Line coverage measures the percentage of source code lines executed by the tests.
- Overall coverage combines the above 2 measures to offer a different, arguably more accurate, view.
- Unit test success percentage, what percentage of the unit tests passed, this should always be close to 100%.

A production ready coverage figure should be 80% or more, i.e. 4/5ths of the code is tested automatically. 100% coverage is usually impractical for non-trivial projects where generated byte-code and environment specific concerns generally get in the way. Code with less than 60% coverage is likely to prove unreliable and require patching if used in production systems. Less than 40% coverage probably means implementing a better test suite or developing from scratch.

Test success percentage should always be checked, 90% test coverage is of little use if 50% or more of the tests fail. Although it's not unusual for projects to occasionally skip a few failing test this should never be more than a few (< 5%) and for a short time.

D.4.4.1.3 Comment coverage

These measures assess the degree to which the code is commented. This is more subjective and does little to measure the quality of the comments. The measures used are:

- Comment density which is just the percentage of comment lines compared to the total number of non-blank lines.
- Public documented API coverage measures the percentage of public classes and methods that are documented.

Comment density alone isn't greatly informative, the average across open source projects is around 30%. Much higher figures aren't guarantee of quality and may simply indicate a verbose commenting style. Public API coverage is more telling. By definition these are the parts of the code that the developer is expecting other developers to use. ALL public methods and classes should be documented, preferably with a standard tool that can be used to create documentation sites, e.g. JavaDoc[4] for Java. This figure should be at 100% for all projects barring very early prototypes. If less than 50% of the public API has been commented it's unlikely to prove the only thing that the developers didn't stay on top of.

[3] https://www.openhub.net/

[4] http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html

D.4.4.1.4 Complexity

Complexity measures, unsurprisingly, give an indication of the complexity of a piece of software. We're using the cyclomatic complexity[5] built into Sonar's Java analysis module. This directly measured the number of paths through the source code and calculates a figure. Every conditional branching statement and return statement increase the Cyclomatic dependency measure. This means that the larger a code base becomes the more the figure increases. To allow comparisons some other measures are calculated:

- Total complexity is the headline figure described above.
- File complexity divides the total by the number of files in the code base.
- Class complexity divides the total by the number of classes in the code base.
- Function complexity divides the total figure by the number of functions in the project.

The three averaged figures allow comparison across projects and indicate the degree to which the complexity has been divided into manageable modules. They're listed in generally decreasing numerical value. This is not guaranteed but as the units are usually aggregated upwards, i.e. functions/methods make up classes and files usually contain at least one class, a dramatic change in ordering would indicate a suspect design. The figure for functions should be less than 5, double figures are usually alarming. The figures for classes and files are more variable but figures over 30 generally indicate over-complex units.

D.4.4.1.5 Technical debt

This quantifies the refactoring effort required to right the wrongs detected by static analysis. There are two measures:

- Effort required in man months.
- Number of issues detected by static analysis, each issue is a assigned a severity indicator: Blocker, Critical, Major, Minor, Info. The ordering is self explanatory.

The measures are provided for comparison but automated effort calculations of this type aren't renowned for their accuracy.

D.4.4.2 Cross project comparison

The figures presented below were taken from recent snapshot builds of the projects, in December 2014.

D.4.4.2.1 Size

The number of lines, files, classes, and functions

	Apache Maven	Apache Tika	Apache PDFBox
Line Count	58467	39933	101894
File Count	677	387	923
Class Count	753	514	979
Function Count	4655	2717	7171
Effort Estimate	87 Years	18 Years	30 Years

The first thing to note is that PDFBox is the largest of the three projects by all measures except estimated effort. Apache Tika is the smallest project of the three according to all measures.

[5] http://en.wikipedia.org/wiki/Cyclomatic_complexity

D.4.4.2.2 Test and comment coverage

These have been grouped together as they're the more objective measures of software quality. All developers should be familiar with the concepts as they're widely recognised as best practise. Projects that have problems here are likely to be struggling on other fronts.

	Apache Maven	Apache Tika	Apache PDFBox
Condition Coverage	31.0%	58.9%	27.9%
Line Coverage	39.3%	64.8%	35.4%
Overall Coverage	36.7%	62.9%	33.3%
% Tests Succeeded	100%	100%	100%
Comment Coverage	21.4%	21.9%	20.1%
Public API Coverage	31.2%	45.8%	86.9%

Apache Tika is the clear leader in terms of test coverage with Maven and PDFBox showing similar results. One issue is that the entire project code base is analysed here. Most projects have core code and this tends to be more thoroughly tested. We'll look again at this when we decompose PDFBox a little more.

D.4.4.2.3 Complexity and technical debt

	Apache Maven	Apache Tika	Apache PDFBox
Total Complexity	10351	9047	8074
File Complexity	15.3	23.4	19.7
Class Complexity	13.7	17.6	18.6
Func Complexity	2.2	3.3	2.5
Number of Issues	1533	9839	8074
Technical Debt	262d	698d	841d

D.4.5 PDFBox Analysis

In this section we'll take a look at the individual PDFBox modules. The figures given in the last section cover all PDFBox modules and are an overall average. In practise VeraPDF won't utilise every PDFBox module for PDF/A validation. The first section describes the PDFBox modules, their function and the internal dependencies between them. It concludes with an assessment of the modules that VeraPDF will depend upon. The following sub-section looks at these modules listing the same measures used in section 4.4 for each.

D.4.5.1 PDFBox Modules

Each PDFBox module is listed along with a brief description.

D.4.5.1.1 PDFBox Reactor

This is the top level Maven project descriptor and builds the other modules. It doesn't carry any code or tests, it does manage dependencies and compilation of the whole project. For purposes of code analysis it's redundant.

D.4.5.1.2 PDFBox Parent

Another Maven project description module the parent pulls in the main apache Maven parent and lists project details. The parent doesn't contain any compilable code and is exempt from static analysis.

D.4.5.1.3 PDFBox Application

This is a bundle module used to create the PDFBox command line application. This is another module that doesn't hold any code and is irrelevant for static analysis purposes.

D.4.5.1.4 Preflight Application

A Maven bundle module for the PDFBox Preflight PDF/A validation command line application. Once again the module doesn't hold any code and will be ignored in our analysis.

D.4.5.1.5 PDFBox

The core PDFBox parsing library and home to many of the important abstractions in the PDFBox code. VeraPDF will certainly depend on the PDFBox module and, by extension, any modules it depends on.

D.4.5.1.6 Fontbox

The PDFBox module that houses the libraries used to obtain low level information from font files. The main PDFBox module depends upon fontbox making it an indirect dependency for veraPDF. It's also possible that veraPDF will depend on fontbox directly initially as developing font parsing modules may not be a primary development priority.

D.4.5.1.7 PDFBox Examples

This is a high level PDFBox module that provides worked examples of how to use the main PDFBox libraries. It's depends upon the PDFBox module and doesn't provide any PDF parsing capabilities. VeraPDF won't depend upon the examples module.

D.4.5.1.8 PDFBox Tools

This module provides command line tools for using PDFBox. It depends upon the main PDFBox libraries but won't used by veraPDF.

D.4.5.1.9 Preflight

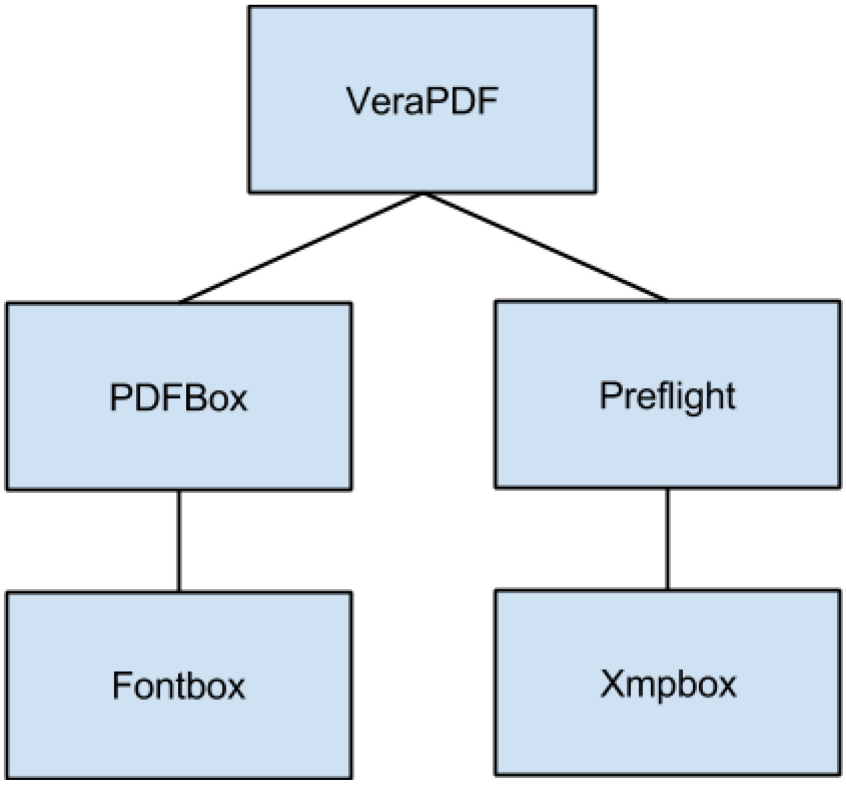
The preflight module holds the libraries for PDF/A level one validation. The module includes low level checks for document characteristics required for validation as well as the validation algorithms. It depends on the PDFBox library and the xmpbox module (see below). VeraPDF will certainly initially depend upon the preflight module for low level checks.

D.4.5.1.10 Xmpbox

Xmpbox is a library for parsing Adobe XMP metadata and it's used by the preflight module for performing the XMP checks required for PDF/A validation. VeraPDF will depend directly upon xmpbox as it's XMP metadata library in addition to any checks called indirectly through the preflight module.

D.4.5.1.11 Summary of veraPDF dependencies

In summary the Apache PDFBox modules and their dependencies exhibit the characteristics of a sensibly designed project. The function of each module is clear and the dependencies between them show a logical and clean structure without circular dependencies. This, in turn, means that the modules required by veraPDF are well defined as shown in the diagram below.



VeraPDF Dependencies

D.4.5.2 Analysis of VeraPDF dependencies

This section presents static code analysis of the PDFBox modules that will be used by VeraPDF as opposed to overall project averages. This is more relevant than the overall figures and gives a better overall picture of the code quality of the modules that veraPDF will depend on.

D.4.5.2.1 Size

Note that there's no effort estimate figures for the individual modules as OpenHub don't provide statistics at any lower level of granularity than project.

	PDFBox	Preflight	Fontbox	XMPBox
Line Count	61391	10774	13904	7067
File Count	501	116	89	72
Class Count	585	120	115	74
Function Count	4768	589	663	728

D.4.5.2.2 Test and comment coverage

Only overall test coverage statistics and number of tests succeeded were readily available and are shown below.

	PDFBox	Preflight	Fontbox	XMPBox
Overall Coverage	42.9%	0%	12.2%	80.6%
% Tests Succeeded	100%	N/A	100%	100%

Xmpbox is the only module to have exemplary unit test coverage while the main PDFBox module's coverage is OK for a project of its size. Coverage for both fontbox and preflight gives cause for concern. Preflight in particular seems to be a problem but closer investigation reveals that it has unit tests that are run in a slightly unconventional manner. Preflight is tested against the Isartor PDF/A test Suite[6] which is downloaded and unpacked from the Maven build. It appears that the execution of these tests isn't picked up by Sonar analysis, hence the 0% coverage figure. EclEmma[7] is an alternative tool that calculates coverage for code run in the Eclipse IDE. Running preflight's Isartor based integration tests using EclEmma reveals that test coverage for the module is actually a much more healthy 62%.

[6] http://www.pdfa.org/2011/08/isartor-test-suite/

[7] http://www.eclemma.org/

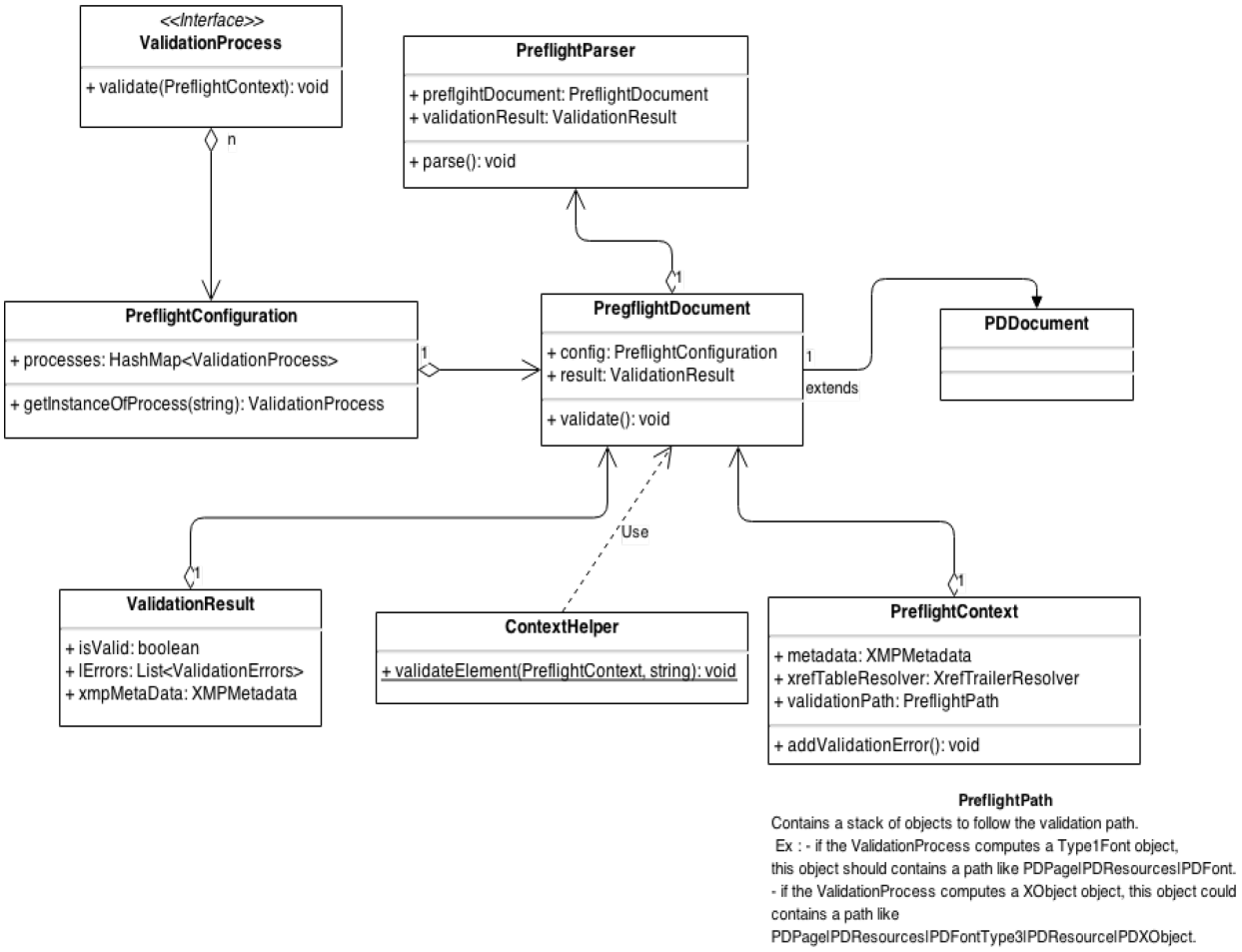
D.4.5.2.3 Complexity and technical debt

	PDFBox	Preflight	Fontbox	XMPBox
Total Complexity	11200	2043	2145	1433
File Complexity	22.4	17.6	24.1	19.9
Class Complexity	19.1	17	18.7	19.4
Func Complexity	2.3	3.7	3.2	2
Number of Issues	6306	409	2767	146
Technical Debt	641	74	200	40

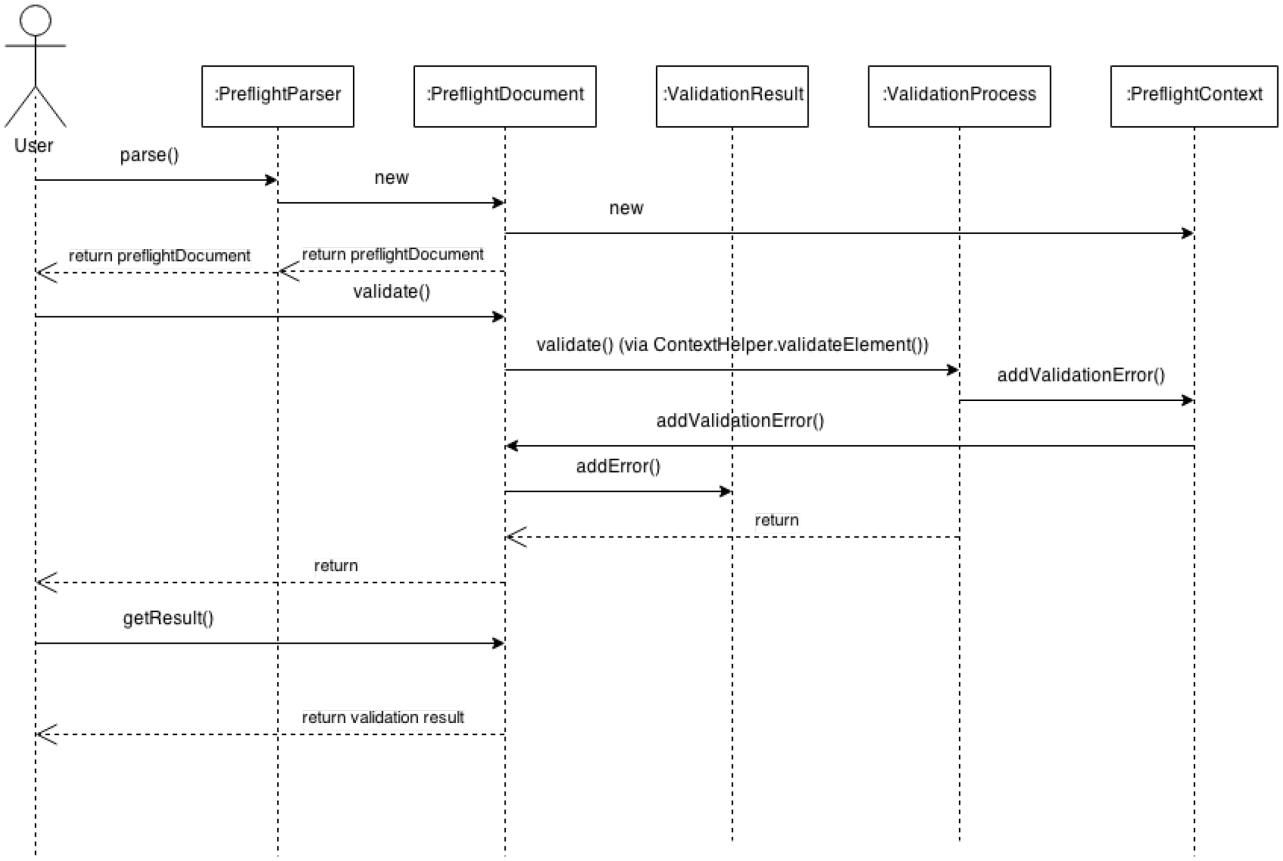
D.5 PDFBox Preflight feasibility study

D.5.1 PDFBox Preflight top level architecture

Class diagram:



Sequence diagram:



D.5.2 PDFBox Preflight analysis

A summary of a study by the Dutch National Library, published in 2009:

- Apache Preflight produces output that is fairly unstructured when used from the command line.
- Apache Preflight was able to identify all documents with encrypted content. However, Preflight doesn't give any specific information on which specific access restrictions apply (e.g. printing, copying, text access).
- The detection of non-embedded fonts turned out to be problematic for Apache Preflight. A simple test file with 1 single font that is not embedded resulted in the following errors : “3.1.3: Invalid Font definition, They are more than one FontFile”. Although Apache Preflight did pick up a font-related issue here, the reported error messages are confusing and do not reflect the actual problem.
- The fact that once a violation of the PDF/A-1b is detected, this may stop Preflight from any further processing of that page.
- Lack of sufficient stability.

Obviously 5 years of development should bring positive changes to the software, so we'll perform tests and codes analysis to check the behavior of Apache Preflight in cases mentioned above:

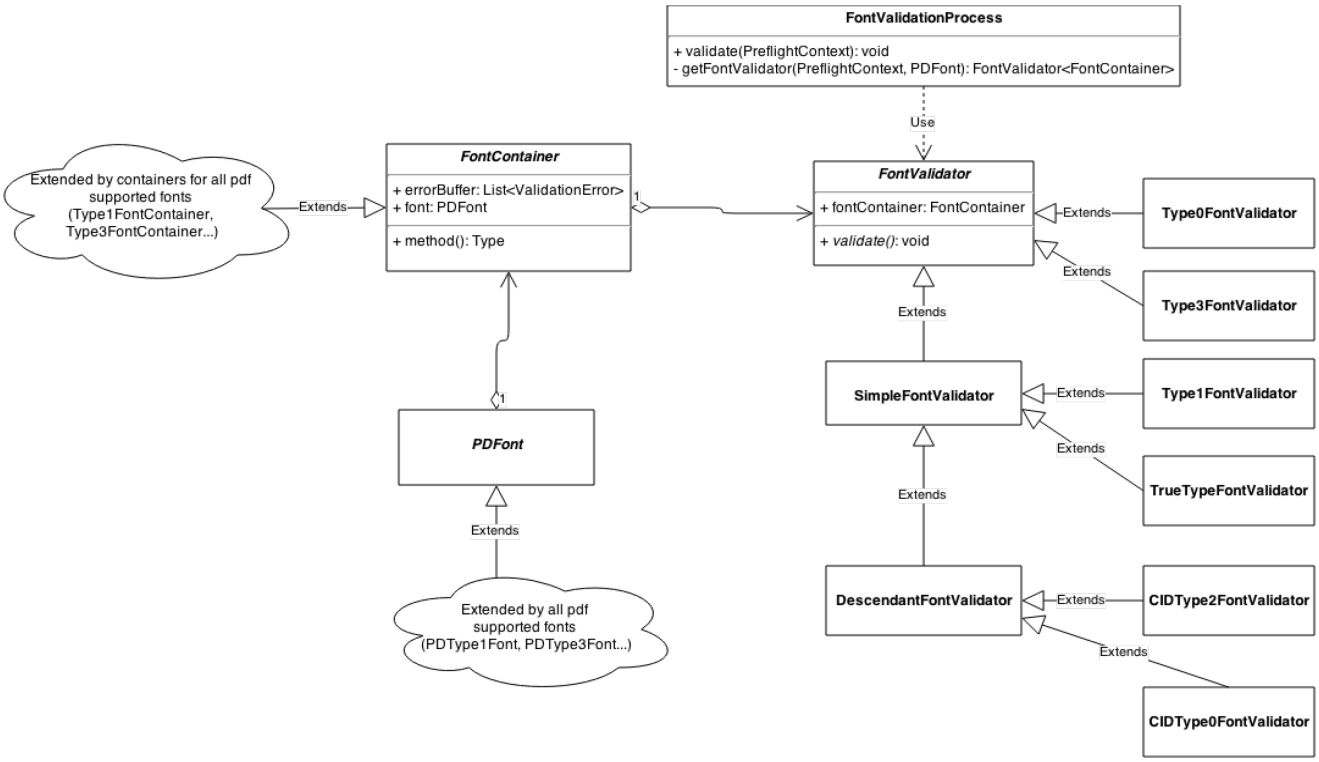
- Since version 2.0 Apache Preflight includes functionality to generate xml reports (org.apache.pdfbox.preflight.parser.XmlResultParser) using the org.w3c.dom package. Here's an example report for a pdf file with invalid trailer :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<preflight name="isartor-6-1-3-t01-fail-a.pdf">
        <executionTimeMS>1966</executionTimeMS>
        <isValid type="PDF/A1-b">false</isValid>
        <errors count="1">
                <error count="1">
                        <code>1.4.1</code>
                        <details>Trailer Syntax error, The trailer dictionary doesn't contain ID</details>
                </error>
        </errors>
</preflight>
```

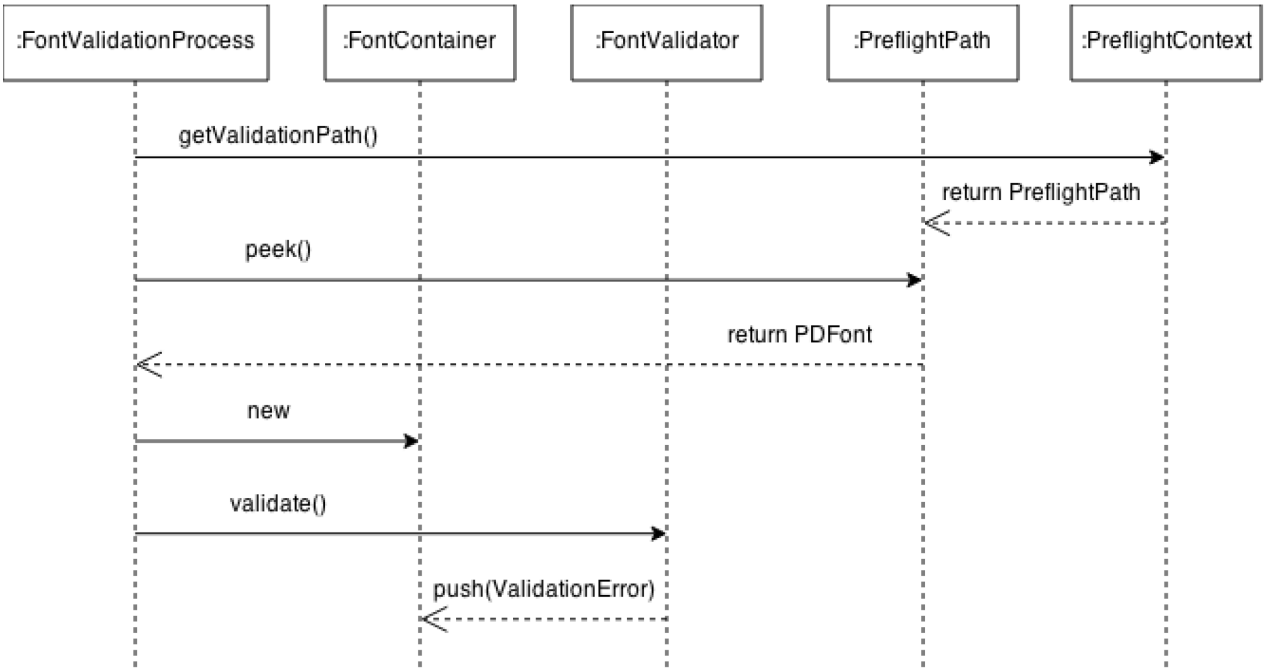
- Encryption validation errors remained unchanged : "Syntax error, Cannot decrypt PDF, the password is incorrect" for documents that require password to open a file and "Trailer Syntax error, The trailer dictionary contains Encrypt" for others (e.g. printing, copying restrictions).
- Font errors detection and reporting were slightly improved:
    - Non-embedded arial true type font : "Invalid Font definition, Arial: FontFile entry is missing from FontDescriptor"
    - PostScript Type 1 font 'LuciduxSans-Oblique' not embedded : "Invalid Font definition, LuciduxSans-Oblique: FontFile entry is missing from FontDescriptor"
    - CID font 'KozMinPro-Regular-Acro' not embedded : "Invalid Font definition, KozMinPro-Regular-Acro: FontFile entry is missing from FontDescriptor"
    - Standard Type 1 font 'Helvetica' not embedded : "Invalid Font definition, Helvetica: FontFile entry is missing from FontDescriptor"
    - font 'Arial' for Form XObject not embedded : "Invalid Font definition, Arial: FontFile entry is missing from FontDescriptor"
    - font 'ZapfDingbats' for field not embedded : "Invalid Font definition, ZapfDingbats: FontFile entry is missing from FontDescriptor"
    - font 'Helvetica' for Type 3 font glyph not embedded : "Font damaged, HelveticaCloneT3.winansi: The Resources dictionary of type 3 font contains invalid font"
    - font 'Arial' for tiling pattern not embedded : "Invalid Font definition, Arial: FontFile entry is missing from FontDescriptor"
- Apache Prefight version 2.0 works with pdf files containing multiple validation problems. Parsing document with 978 pages (PDF 1.4 spec) shows a huge list of validation errors.
- Apache Preflight version 2.0 seems to be pretty stable. It successfully parses documents containing one thousand pages. Will test it on synthetic documents containing 100.000 - 3.000.000 pages in next few days.

D.5.3 PDFBox Preflight font validation

Class diagram:



Sequence diagram:



The principle PDF/A requirements for embedded font files are:

1. The embedded font shall contain the description of all glyphs used in the PDF document.
2. Widths information for these glyphs shall be consistent between the embedded font program and the font dictionary in the PDF document.

PDFBox implements these two checks for all file types except for OpenType (not permitted in PDF/A-1 documents, but allowed in PDF/A-2 ad PDF/A-3):

Font type	Incorrect width check	Missing glyph check	Parsing code
CIDType0	Success	Success	org.apache.fontbox.cff.CFFParser, org.apache.fontbox.cmap.CMapParser
CIDType2	Success	Success	org.apache.fontbox.ttf.TTFParser, org.apache.fontbox.cmap.CMapParser
Type1	Success	Success	org.apache.fontbox.type1.Type1Parser
Type1 CFF	Success	Success	org.apache.fontbox.cff.CFFParser
TrueType	Success	Success	org.apache.fontbox.ttf.TTFParser
OpenType	?	?	org.apache.fontbox.ttf.OTFParser (isn't implemented yet)

D.6 XMPBox

D.6.1 Supported XMP schemas

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
Dublin Core Schema	dc:contributor	bag ProperName	...DublinCoreSchema.CONTRIBUTOR	Text	Bag
	dc:coverage	Text	...DublinCoreSchema.COVERAGE	Text	Simple
	dc:creator	seq ProperName	...DublinCoreSchema.CREATOR	Text	Seq
	dc:date	seq Date	...DublinCoreSchema.DATE	Date	Seq
	dc:description	Lang Alt	...DublinCoreSchema.DESCRIPTION	LangAlt	Simple
	dc:format	MIMEType	...DublinCoreSchema.FORMAT	MIMEType	Simple

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
Dublin Core Schema	dc:identifier	Text	...DublinCoreSchema.IDENTIFIER	Text	Simple
	dc:language	bag Locale	...DublinCoreSchema.LANGUAGE	Text	Bag
	dc:publisher	bag ProperName	...DublinCoreSchema.PUBLISHER	Text	Bag
	dc:relation	bag Text	...DublinCoreSchema.RELATION	Text	Bag
	dc:rights	Lang Alt	...DublinCoreSchema.RIGHTS	LangAlt	Simple
	dc:source	Text	...DublinCoreSchema.SOURCE	Text	Simple
	dc:subject	bag Text	...DublinCoreSchema.SUBJECT	Text	Bag
	dc:title	Lang Alt	...DublinCoreSchema.TITLE	LangAlt	Simple
	dc:type	bag open Choice	...DublinCoreSchema.TYPE	Text	Bag
XMP Basic Schema	xmp:Advisory	bag XPath	...XMPBasicSchema.ADVISORY	XPath	Bag
	xmp:BaseURL	URL	...XMPBasicSchema.BASEURL	URL	Simple
	xmp:CreateDate	Date	...XMPBasicSchema.CREATEDATE	Date	Simple
	xmp:CreatorTool	AgentName	...XMPBasicSchema.CREATORTOOL	AgentName	Simple
	xmp:Identifier	bag Text	...XMPBasicSchema.IDENTIFIER	Text	Bag
	xmp:MetadataDate	Date	...XMPBasicSchema.METADATADATE	Date	Simple
	xmp:ModifyDate	Date	...XMPBasicSchema.MODIFYDATE	Date	Simple
	xmp:Nickname	Text	...XMPBasicSchema.NICKNAME	Text	Simple
	xmp:Thumbnails	alt Thumbnail	...XMPBasicSchema.THUMBNAILS	Thumbnail	Alt
	xmp:Label	Text	...XMPBasicSchema.LABEL	Text	Simple

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
XMP Basic Schema	xmp:Rating	Closed Choice of Integer	...XMPBasicSchema.RATING	Integer	Simple
	n/a		...XMPBasicSchema.MODIFIER_DATE	Date	Simple
XMP Rights Management Schema	xmpRights:Certificate	URL	...XMPRightsManagementSchema.CERTIFICATE	URL	Simple
	xmpRights:Marked	Boolean	...XMPRightsManagementSchema.MARKED	Boolean	Simple
	xmpRights:Owner	bag ProperName	...XMPRightsManagementSchema.OWNER	ProperName	Bag
	xmpRights:UsageTerms	Lang Alt	...XMPRightsManagementSchema.USAGETERMS	LangAlt	Simple
	xmpRights:WebStatement	URL	...XMPRightsManagementSchema.WEBSTATEMENT	URL	Simple
XMP Media Management Schema	xmpMM:DerivedFrom	ResourceRef	...XMPMediaManagementSchema.DERIVED_FROM	ResourceRef	Simple
	xmpMM:DocumentID	URI	...XMPMediaManagementSchema.DOCUMENTID	URI	Simple
	xmpMM:History	seq ResourceEvent	...XMPMediaManagementSchema.HISTORY	ResourceEvent	Seq
	xmpMM:LastURL	- (deprecated)	...XMPMediaManagementSchema.LAST_URL	URL	Simple
	xmpMM:ManagedFrom	ResourceRef	...XMPMediaManagementSchema.MANAGED_FROM	ResourceRef	Simple
	xmpMM:Manager	AgentName	...XMPMediaManagementSchema.MANAGER	AgentName	Simple
	xmpMM:ManagerVariant	Text	...XMPMediaManagementSchema.MANAGERVARIANT	Text	Simple
	xmpMM:ManageTo	URI	...XMPMediaManagementSchema.MANAGETO	URI	Simple

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
XMP Media Management Schema	xmpMM:ManageUI	URI	...XMPMediaManagementSchema.MANAGEUI	URI	Simple
	xmpMM:RenditionClass	RenditionClass	...XMPMediaManagementSchema.RENDITIONCLASS	RenditionClass	Simple
	xmpMM:RenditionOf	- (deprecated)	...XMPMediaManagementSchema.RENDITION_OF	ResourceRef	Simple
	xmpMM:RenditionParams	Text	...XMPMediaManagementSchema.RENDITIONPARAMS	Text	Simple
	xmpMM:SaveID	- (deprecated)	...XMPMediaManagementSchema.SAVE_ID	Integer	Simple
	xmpMM:VersionID	Text	...XMPMediaManagementSchema.VERSIONID	Text	Simple
	xmpMM:Versions	seq Version	...XMPMediaManagementSchema.VERSIONS	Version	Seq
	xmpMM:InstanceID	URI	...XMPMediaManagementSchema.INSTANCEID	URI	Simple
	n/a		...XMPMediaManagementSchema.ORIGINALDOCUMENTID	Text	Simple
	n/a		...XMPMediaManagementSchema.INGREDIENTS	Text	Bag
XMP Basic Job Ticket Schema	xmpBJ:JobRef	bag Job	...XMPBasicJobTicketSchema.JOB_REF	Job	Bag
XMP Paged-Text Schema	xmpTPg:MaxPageSize	Dimensions	...XMPageTextSchema.MAX_PAGE_SIZE	Dimensions	-
	xmpTPg:NPages	Integer	...XMPageTextSchema.N_PAGES	Integer	-
Adobe PDF Schema	pdf:Keywords	Text	...AdobePDFSchema.KEYWORDS	Text	Simple
	pdf:PDFVersion	Text	...AdobePDFSchema.PDF_VERSION	Text	Simple
	pdf:Producer	AgentName	...AdobePDFSchema.PRODUCER	Text	Simple

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
Photoshop Schema	photoshop:AuthorsPosition	Text	...PhotoshopSchema.AUTHORS_POSITION	Text	Simple
	photoshop:CaptionWriter	ProperName	...PhotoshopSchema.CAPTION_WRITER	ProperName	Simple
	photoshop:Category	Text	...PhotoshopSchema.CATEGORY	Text	Simple
	photoshop:City	Text	...PhotoshopSchema.CITY	Text	Simple
	photoshop:Country	Text	...PhotoshopSchema.COUNTRY	Text	Simple
	photoshop:Credit	Text	...PhotoshopSchema.CREDIT	Text	Simple
	photoshop:DateCreated	Date	...PhotoshopSchema.DATE_CREATED	Date	Simple
	photoshop:Headline	Text	...PhotoshopSchema.HEADLINE	Text	Simple
	photoshop:Instructions	Text	...PhotoshopSchema.INSTRUCTIONS	Text	Simple
	photoshop:Source	Text	...PhotoshopSchema.SOURCE	Text	Simple
	photoshop:State	Text	...PhotoshopSchema.STATE	Text	Simple
	photoshop:SupplementalCategories	Text	...PhotoshopSchema.SUPPLEMENTAL_CATEGORIES	Text	Bag
	photoshop:TransmissionReference	Text	...PhotoshopSchema.TRANSMISSION_REFERENCE	Text	Simple
	photoshop:Urgency	Integer	...PhotoshopSchema.URGENCY	Integer	Simple
	n/a		...PhotoshopSchema.ANCESTORID	URI	Simple
	n/a		...PhotoshopSchema.COLOR_MODE	Integer	Simple
	n/a		...PhotoshopSchema.DOCUMENT_ANCESTORS	Text	Bag

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
Photoshop Schema	photoshop:History		...PhotoshopSchema.HISTORY	Text	Simple
	n/a		...PhotoshopSchema.ICC_PROFILE	Text	Simple
	n/a		...PhotoshopSchema.TEXT_LAYERS	Layer	Seq
Exiff Schema for TIFF Properties	tiff:Artist	ProperName	...TiffSchema.ARTIST	ProperName	Simple
	tiff:BitsPerSample	seq Integer	...TiffSchema.BITS_PER_SAMPLE	Integer	Seq
	tiff:Compression	Closed Choice of Integer	...TiffSchema.COMPRESSION	Integer	Simple
	tiff:Copyright	Lang Alt	...TiffSchema.COPYRIGHT	LangAlt	Simple
	tiff:DateTime	Date	...TiffSchema.DATE_TIME	Date	Simple
	tiff:ImageDescription	Lang Alt	...TiffSchema.IMAGE_DESCRIPTION	LangAlt	Simple
	tiff:ImageLength	Integer	...TiffSchema.IMAGE_LENGHT	Integer	Simple
	tiff:ImageWidth	Integer	...TiffSchema.IMAGE_WIDTH	Integer	Simple
	tiff:Make	ProperName	...TiffSchema.MAKE	ProperName	Simple
	tiff:Model	ProperName	...TiffSchema.MODEL	ProperName	Simple
	tiff::Orientation	Closed Choice of Integer	...TiffSchema.ORIENTATION	Integer	Simple
	tiff:PhotometricInterpretation	Closed Choice of Integer	...TiffSchema.PHOTOMETRIC_INTERPRETATION	Integer	Simple
	tiff:PlanarConfiguration	Closed Choice of Integer	...TiffSchema.PLANAR_CONFIGURATION	Integer	Simple
	tiff:PrimaryChromaticities	seq Rational	...TiffSchema.PRIMARY_CHROMATICITIES	Rational	Seq

PDF allowed XMP schemas	Schema properties	Value Type	XMPBox class field (org.apache.xmpbox.schema.)	Preflight value type (org.apache.xmpbox.type.Types)	Cardinality (org.apache.xmpbox.type.Cardinality)
Exiff Schema for TIFF Properties	tiff:ReferenceBlackWhite	seq Rational	...TiffSchema.REFERENCE_BLACK_WHITE	Rational	Seq
	tiff:ResolutionUnit	Closed Choice of Integer	...TiffSchema.RESOLUTION_UNIT	Integer	Simple
	tiff:SamplesPerPixel	Integer	...TiffSchema.SAMPLES_PER_PIXEL	Integer	Simple
	tiff:Software	AgentName	...TiffSchema.SOFTWARE	AgentName	Simple
	tiff:TransferFunction	seq Integer	...TiffSchema.TRANSFER_FUNCTION	Integer	Seq
	tiff:WhitePoint	seq Rational	...TiffSchema.WHITE_POINT	Rational	Seq
	tiff:XResolution	Rational	...TiffSchema.XResolution	Rational	Simple
	tiff:YCbCrCoefficients	seq Rational	...TiffSchema.YCB_CR_COEFFICIENTS	Rational	Seq
	tiff:YCbCrPositioning	Closed Choice of Integer	...TiffSchema.YCB_CR_POSITIONING	Integer	Seq
	tiff:YCbCrSubSampling	Closed Choice of seq Integer	...TiffSchema.YCB_CR_SUB_SAMPLING	Integer	Simple
	tiff:YResolution	Rational	...TiffSchema.YRESOLUTION	Rational	Simple
EXIF Schema for EXIF-specific Properties			...TiffSchema		
PDF/A Extension Schema *Container Schema*	pdfaExtension:schemas	bag Schema	...PDFAExtensionSchema.SCHEMAS	PDFASchema	Bag

D.7 PDF Document I/O

D.7.1 PDF stream assumptions in PDFBox

The PDF stream access in PDFBox is implemented via the class PushBackInputStream .

PushBackInputStream is a binary stream an extension of java.io.PushbackInputStream, that adds functionality for seeking through the PDF input stream and some little features simplifying parsing PDF (e.g. peek method that allows to read next byte from stream but keep current offset position untouched) document. Supported operations include seek, and we can add support for mark and reset operations. The strategy to deal with non-seekable streams will be discussed.

D.7.2 PDF parser

PDFBox contains two different implementations of PDFParser:

- PDFParser doesn't assume that PDF Document complies to PDF 1.7 or ISO 32000-1:2008 specification and parses PDF Document sequentially, ignoring contents of xref table.
- NonSequentialPDFParser is more recent than PDF Parser and assumes that the PDF Document has a low level object structure conformant to PDF.1.7 (ISO 32000-1:2008), so this parser will be used during validation.

D.8 Supported stream filters

Name	PDFBox class	PDFBox support	Supported decode params	PDF/A-1	PDF/A-2	PDF/A-3
ASCIIHexDecode	org.apache.pdfbox.filter.ASCII85Filter	Decode and encode	No params available			
ASCII85Decode	org.apache.pdfbox.filter.ASCIIHexFilter	Decode and encode	No params available			
LZWDecode	org.apache.pdfbox.filter.LZWFilter	Decode and encode	Predictor, Colors (complete support)	Not permitted	Not permitted	Not permitted
FlateDecode	org.apache.pdfbox.filter.FlateFilter	Decode and encode	Predictor, Colors (complete support)			
RunLengthDecode	org.apache.pdfbox.filter.RunLengthDecodeFilter	Only decode supported	No params available			
CCITTFaxDecode	org.apache.pdfbox.filter.CCITTFaxDecodeFilter	Only decode supported	K, EncodedByteAlign, Columns, Rows, BlackIs1 (partially support. Unsupported params - EndOfLine, EndOfBlock, DamagedRowsBeforeError)			
DCTDecode	org.apache.pdfbox.filter.DCTFilter	Isn't implemented yet (see 7.4.8)				
JBIG2Decode	org.apache.pdfbox.filter.JBIG2Filter	Only decode supported	JBIG2Globals (complete support)			
JPXDecode	org.apache.pdfbox.filter.JPXFilter	Only decode supported	No params available	Not applicable		

Name	PDFBox class	PDFBox support	Supported decode params	PDF/A-1	PDF/A-2	PDF/A-3
Crypt	org.apache.pdfbox.filter.CryptFilter	Decode and encode (Only identity crypt filter)	Name (partially support. Unsupported params - Type)	Not applicable	Permitted[8]	Permitted[9]

Note that support levels of all the filters above were also verified by checking the source code

[8] when the when the value of the "Name" key in the decode parameters dictionary is "Identity"
[9] when the value of the "Name" key in the decode parameters dictionary is "Identity".

Annex E: License Compatibility

E.1 Introduction

This document describes the dependencies and licensing implications of the veraPDF functional and technical designs. It demonstrates compatibility with the required open source licenses.

Section 2 describes the licensing requirements relating to software and test corpora. Section 3 describes the rationale for using existing software to deliver certain aspects of the Conformance Checker functionality. Section 4 describes the scenarios for each Conformance Checker component under which we propose to reuse or improve existing software. Section 5 refers to authoritative sources to demonstrate compatibility between commonly used open source licenses and the required licenses. Section 6 identifies dependencies in the veraPDF designs and demonstrates the necessary license compatibility to permit the proposed use.

E.2 PREFORMA requirements

Software

In the Framework Agreement[10] PREFORMA requires all software developed during the project to be licensed under two specific open source licenses (the 'PREFORMA licenses'):

- GNU General Public License v3 or later (GPLv3+);
- Mozilla Public License v2 or later (MPLv2+).

In the clarification[11] issued on 26 February 2015, PREFORMA confirmed the interpretation of licensing compatibility:

(PREFORMA Requirement 1)

1. All code (software and libraries) distributed as part of the Conformance Checker, either developed during the project or already developed by the supplier and contributed to PREFORMA, is to be released under GPLv3++ and MPLv2++.

(PREFORMA Requirement 2)

2. All code (software and libraries) distributed as part of the Conformance Checker, either developed during the project or already developed by a third party and contributed by the supplier to PREFORMA, has to be freely available in open source form under generally recognized free software licenses compatible with the GPLv3++ and MPLv2++ to enable redistribution of the whole package under these two licenses.
   All code (software and libraries) required to compile and/or execute the Conformance Checker in a production environment has to be freely available in open source form under generally recognized free software licenses compatible with the GPLv3++ and MPLv2++ to enable redistribution of the whole package under these two licenses.

E.3 Rationale for reusing existing software

The stated aims of the open source approach include an intention for us (the contractors) "to be active contributors in other relevant Open Source projects that are related to the Open Source project for which [we] are contracted"[12]. We interpret this as a desire on the part of the PREFORMA consortium to contribute to a healthy open source ecosystem of active software projects, by reusing and even improving existing open source software for the purposes of building the Conformance Checker within the expectations described above.

Reuse and improvement of existing software is both a common practice in open source development and the most efficient way to deliver the Conformance Checker functionality without reinventing wheels when existing software can be built upon. Reusing rather than reinventing has two distinct benefits:

1. greater **reliability**: which comes from exposure to testing over a long period of time and an existing community of maintainers;
2. greater **efficiency**: by focusing scarce development resources on functionality which is core to the Conformance Checker and not currently available in existing software.

[10] PREFORMA Framework Agreement v1.0 (section 17.3, p. 15)

[11] PREFORMA_clarification_from_PREFORMA_on_licensing_requirements.pdf (received via email)

[12] PREFORMA Invitation to Tender v1.0 (p. 15)

Community contributions

In building the open source community that will sustain veraPDF after the funded period, it will be necessary to accept third-party contributions to the project. The legal arrangements (for example contributor agreements) for accepting contributions such that they align with the licensing requirements and do not affect the licensing of the Conformance Checker are defined in CE 3.3 Code.

E.4 Scenarios for reusing existing software

The original veraPDF Tender Proposal, section V *Technical Approach* (p. 22) presented two options for development:

- building on existing open-source tools;
- developing a 'greenfield' solution (an entirely new codebase, from scratch).

PREFORMA further explain that:

> The use of third party code under other open source licenses should be an instrument enabling suppliers to devote the maximum of their resources to the developement of new code that is required by the Conformance Checker but does not yet exist. There is a risk however that an overly dependancy on third party code for realising the Conformance Checker could cause the project to end up as a mashup of existing software with little innovation value.

In consideration of the following...

> It follows that the possibilities for use of third party code for the core functionality, i.e. the implementation checker and the policy checker, is rather limited. In particular for the implementation checker, where the use of third party code has to be considered only under extraordinary circumstances and would require the explicit consent of the PREFORMA Consortium. The use of third party code for the subsidiary functions, i.e. shell, reporter, and metadata fixer, which are not the central objective of PREFORMA, could however be considered with a less restrictive view.

...we will pursue a nuanced approach, ensuring that the Implementation Checker and the Metadata Fixer rely for their core functionality on entirely greenfields solutions which will be licensed under GPLv3+/MPLv2+ (Requirement 1). For the Policy Checker, Reporter, and Shell we propose the use of third-party software such that generic functionality is provided by established and robust tools compatible with GPLv3+/MPLv2+ (Requirement 2).

This will meet the stated aim of devoting funding to innovation and developing new format validation functionality that is not currently available in existing software.

E.4.1 Implementation Checker

The Implementation Checker, along with other components of the Conformance Checker, rely on a PDF Parser, as described in FS 3.1.1.1 PDF Parsers. We have considered the use of PDFBox in detail, as described in Annex D: PDFBox Feasibility Study.

In summary, PDFBox provides a PDF Parser as well as (limited) PDF/A Validation functionality. In light of the requirement to ensure that the Implementation Checker will be entirely licensed under GPLv3+/MPLv2+ we will:

- develop a greenfield Implementation Checker not using PDFBox PDF/A Validation;
- develop a greenfield PDF Parser not using the PDFBox PDF Parser.

Indicative costings for this approach were provided in the the original veraPDF Tender Proposal. This is in line with the Evaluation Report[13] which informed us that:

> Only the greenfield solution using "GPLv3 or later and MPLv2 or later" would meet the minimum requirements in PREFORMA

In order to progress development during the first half of Phase 2 we are proposing the use of the PDFBox PDF Parser as a reference implementation so that development of the Implementation Checker can begin immediately and be subject to testing for the maximum amount of time available. During the Phase 2 redesign, our greenfield PDF Parser will be swapped with the PDFBox PDF Parser so that the final Conformance Checker prototype delivered at the end of Phase 2 will include an Implementation Checker which is entirely GPLv3+/MPLv2+ for its core functionality. See Annex B. Technical Milestones and Deliverables for more detail.

E.4.2 Metadata Fixer

The Metadata Fixer relies on a PDF Writer to output a Repaired PDF Document containing changes to its PDF Metadata. veraPDF will develop a greenfields PDF Writer so that the Metadata Fixer will be entirely licensed under GPLv3+/MPLv2+.

As with the Implementation Checker, in order to progress development during the first half of Phase 2 we are proposing the use of the PDFBox as a reference implementation so that development of the Metadata Fixer can begin immediately and be subject to testing for the maximum amount of time available. During the Phase 2 redesign, our greenfield PDF Writer will be swapped with the PDFBox component so that the final Conformance Checker prototype delivered at the end of Phase 2 will include a Metadata Fixer which is entirely GPLv3+/MPLv2+ for its core functionality.

E.4.3 Policy Checker

As designed, the Policy Checker relies on the PDF Features Report generated by the Implementation Checker using the PDF Parser. In order to enforce rules on the PDF Features Report we have proposed the use of Schematron for Policy Profiles as described in TS 5 Policy Profile.

Schematron is an open standard with several open source libraries available. It is also possible to handle Schematron documents (Policy Profiles) using generic XML/XSLT libraries such as those available within the Java JDK, however Schematron libraries provide a more straightforward and reliable way to handle the enforcement of Policy Checks.

All proposed dependencies are components operating on open standards and available under open source licenses compatible with GPLv3+/MPLv2+.

E.4.4 Reporter

The Reporter transforms Machine-readable and Human-readable Reports (see FS 2.4 veraPDF Reporter) using generic components for handling open formats such as XML/HTML/PDF. We propose the use of Xalan for XML/XSLT however libraries such as those available within the Java JDK could be used instead. We propose the use of Apache FOP for formatting Human-readable Reports in PDF.

Internationalisation (see TS 7 Internationalization) will use Translation Memory eXchange (TMX) for the Language Packs providing translation between languages. TMX is an open standard and we propose the use of an open source validator and editor.

[13] PREFORMA Evaluation Report (20141003_05_VeraConsortium_consolidated_v1.0.pdf, received via email 06/10/14, p. 2)

All proposed dependencies are generic components operating on open standards and available under open source licenses compatible with GPLv3+/MPLv2+. Users are also able to provide Report Templates transforming Machine-readable or Human-readable Reports into any format of their choosing (see TS 8 Report Template format).

E.4.5 Shell

The Shell manages interactions with users through the interfaces described in FS 4 Interfaces.

Our proposals are:

- FS 4.1.1 Command Line Interface (CLI): to use Apache CLI for parsing command line parameters;
- FS 4.1.2 Desktop Graphical User Interface (GUI-D): to use the same framework as the Web GUI;
- FS 4.2.2 Web Graphical User Interface (GUI-W): to use JQuery and perhaps Bootstrap.

In addition, the Web GUI depends on the REST API:

- TS 1.6.3.5 REST API: to use the JavaX.RS REST Services which are available within the Java JDK and perhaps other frameworks such as DropWizard;

All proposed dependencies are generic web components available under licenses compatible with GPLv3+/MPLv2+.

E.4.6 High level dependencies

These are divided into two categories:

- the development toolset includes software products used in the course of development (generally to edit, build, and test code but the definition could also include editors used to create documentation);
- supporting services include online services used during the project to support development or facilitate cooperation (for example to host source code repositories, provide continuous integration, or track issues).

E.4.6.1 Development toolset

The choice of development tools relies on underlying technology, for example the chosen development language. PREFORMA requires that veraPDF "must be built for portability between technical deployment platforms. (platform independent)"[14] leading to a choice between development in Java or C++. Other tools are used by us during the project but are not required to "develop, maintain, test, and operate" the software - users can choose their own alternatives without affecting the software functionality.

E.4.6.1.1 Development language

Both alternatives (Java and C++) place dependencies on existing software, in order to compile and execute the veraPDF software. The Evaluation Report states that the "use of Java causes a dependency on the Java Virtual Machine" - this is discussed fully below - while the alternative choice of C++ would place requirements on compilers for different platforms.

[14] PREFORMA Invitation to Tender v1.0 (section 5.1, p. 12)

For Java, the dependencies include:

- A Java Development Kit (JDK) required to compile Java code to bytecode (this also provides the native Java software libraries which provide low-level functionality such as file I/O);
- A Java virtual machine (JVM) required to execute the Java bytecode in the user's environment (desktop or server).

The JDK is used to develop and compile the code and is only needed if you want to change the software (presuming a compiled version is freely available, which is required for various platforms). There are fewer JDKs available with only two in wide use:

- the official Oracle JDK (Oracle own the Java trademark) which is GPL licensed (this also includes the official HotSpot JVM which is GPLv2 licensed);
- the OpenJDK, which is the official JavaSE 7 reference implementation and is available under the GPL (this also includes IcedTea JVM, the most popular HotSpot alternative).

The JVM is required by anyone who wants to operate the software and there are a large number of JVMs available in free and open source implementations.[15]

In principle these are all that are required to compile and run all Java software and the software won't be tied to a particular JDK or JVM implementation and will be tested on both of the widely used JVMs. Note that it is not possible to control which JVM a user chooses to use when executing the software however both the main alternatives comply with the requirement to be compatible with GPLv3+/MPLv2+ (Requirement 2).

E.4.6.1.2 Other tools

In practice, developers don't edit code with plain text editors or call the Java compiler directly on the command line to build it. It's time consuming and becomes impractical for projects with more than a handful of source files. Most software developers use a trusted set of tools to organise, edit and build software projects and the choice reflects personal preferences. This is true for the tools we've chosen but there is no absolute requirement to use any of them in order to alter, build or execute the source code.

We intend to use Git for source code management, technically known as revision control. This was chosen as it's used by 50% of the worlds open source projects, including the Linux kernel and is licensed under the GPL. Git isn't required to access the latest version of the code or to build it. Only people wanting to access the source history (who changed what and when) would need to install Git, or visit the project's GitHub site where it's available online. All committers to the projects source will have to use Git. Committers are individuals authorised to make direct changes to the official veraPDF git repository. This doesn't include contributions which are submitted as a patch which is then tested and applied by a committer.

In order to build the Java project and manage software dependencies we'll use Maven, the Apache software lifecycle tool. This will also be used to generate JavaDoc documentation, a source code information website and package software releases. Maven is available under the Apache 2 license. The veraPDF source code will include the Maven POM.xml projects files which provide the tool with structured information about the tools used to build the project and its dependencies. This is simply for convenience so that all developers who use Maven can build the source with a single command. Also most Java development environments support the POM format and can automatically import Maven projects. It's possible to build and run Maven projects without ever using Maven although a little initial effort would be required to set up an alternative build system.

[15] http://en.wikipedia.org/wiki/List_of_Java_virtual_machines

Finally we'll be using Jenkins as the official project continuous integration server available under the MIT license. This is a supporting tool and there's no need to install or use Jenkins at all, although anyone is free to visit the web GUI of the server[16].

We've deliberately not covered Integrated Development Environments (IDEs) because we won't be mandating one. Developers are free to use any tool that supports Java development, indeed it's likely that individual developers within the consortium will use different tools. To be clear an IDE license doesn't usually have any implications for the code developed using it.

In summary we are confident that we'll be using only open source licensed tools to develop the software. Furthermore we're decoupled from any specific implementation and credible open source alternatives are also available.

E.4.6.2 Supporting Services

Running veraPDF as a genuine open source project introduces specific requirements. In order to establish an open source community we'll need to be visible and accessible online as early as possible. To this end we'll be using established third parties that provide more robust and reliable services that we could build. Use of these services represent a further level of decoupling from the delivered software and licensing shouldn't be an issue. In most cases they're based on open source software but may use proprietary code in places.

We'll be using GitHub to host our source code repository online, allowing other developers to easily access and clone it. It also provides other services such as issue tracking, online editing and a web GUI that ties it all together. The service is based on Git (GPL) but with proprietary code to distinguish them from the competition, e.g. the online source code editor. GitHub are committed to providing free hosting to all open source projects on an ongoing basis and it has become the de-facto home to the majority of such software with over 10 million source repositories. Licensing not an issue as only Git, the supporting tool, and its GPL license have any project ramifications.

It's likely that we will use Travis, an online Continuous Integrations service, that is integrated with GitHub. Travis provides vanilla Ubuntu VMs for building software, and supports testing against both widely used JDKs and JVMs. Again it is possible that some non-open source software is used by Travis but this does not have licensing implications for the veraPDF software. Again they are committed to providing a free to use service for open source projects.

In conclusion, all proposed services are free to use but not necessarily completely open source licensed. These are only supporting services which are very loosely coupled to the software itself and any of them could be replaced easily.

[16] http://jenkins.opf-labs.org/

E.5 Legal analysis

E.5.1 Scenarios requiring license compatibility

The proposed uses of existing software create two scenarios requiring licensing compatibility.

Scenario	**Description**	**Licensing requirement**
Reuse	'Linking' with the existing software to provide specific functionality to veraPDF. Linking can also require 'redistributing' with the final, packaged product however use for 'testing' during development or building the software does not require redistribution.	That the licenses are "compatible with" GPLv3+/MPLv2+ as described in Requirement 2. See Table 2.
Improvement	Submitting source code changes back to the existing software (for example patching bugs or enhancing functionality).	That the contributed code complies with the license of the existing software. As veraPDF retain the copyright in code developed during PREFORMA we will license the contributions as required by the existing software.

Table 1: *Scenarios requiring licensing compatibility*

E.5.2 Open source license compatibility

To demonstrate license compatibility we refer to authoritative sources to show that common open source licenses are "compatible with" GPLv3+/MPLv2+.

ID	Original license	Analysis	Source
1	Apache 2.0	"Apache 2 software can ... be included in GPLv3 projects"	Apache Software Foundation[17]
		"[Apache 2.0] is ... compatible with version 3 of the GNU GPL"	Free Software Foundation[18]
		"May I combine MPL-licensed code and [Apache]-licensed code in the same executable program? … Yes"	Mozilla Foundation[19]

[17] http://www.apache.org/licenses/GPL-compatibility.html

[18] https://www.gnu.org/licenses/license-list.html#apache2

[19] https://www.mozilla.org/MPL/2.0/FAQ.html

ID	Original license	Analysis	Source
2	LGPL 2.1 or 3	“Compatible with … GPL v3”	Free Software Foundation[20]
		“May I combine MPL-licensed code and (L)GPL-licensed code in the same executable program? … Yes”	Mozilla Foundation[21]
3	BSD	“Redistribution and use in source and binary forms, with or without modification, are permitted”	Open Source Initiative[22, 23]
4	MIT	“Permission is ... granted, free of charge ... to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense”	Open Source Initiative[24]
5	Common Development and Distribution (CDDL v1.1)	“You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice”	Open Source Initiative[25]
6	Eclipse Public License 1.0	“each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form”	Eclipse Foundation[26]

[20] https://www.gnu.org/licenses/license-list.html#LGPL

[21] https://www.mozilla.org/MPL/2.0/FAQ.html

[22] http://opensource.org/licenses/BSD-2-Clause

[23] http://opensource.org/licenses/BSD-3-Clause

[24] http://opensource.org/licenses/MIT

[25] http://opensource.org/licenses/CDDL-1.0

[26] https://www.eclipse.org/legal/epl-v10.html

ID	Original license	Analysis	Source
7	GPLv2	“GPLv2 is, by itself, not compatible with GPLv3. However, most software released under GPLv2 allows you to use the terms of later versions of the GPL as well. When this is the case, you can use the code under GPLv3 to make the desired combination.”	Free Software Foundation[27]
		“Section 3.3 provides indirect compatibility between [MPL v2.0] and the GNU GPL version 2.0”	Free Software Foundation[28]
8	Affero GPL v3	“[AGPL v3] is ... technically not compatible with GPLv3 in a strict sense … However, you are allowed to combine separate modules or source files released under both of those licenses in a single project	Free Software Foundation[29]
		“Section 3.3 provides indirect compatibility between [MPL v2.0] and … the GNU AGPL version 3.0”	Free Software Foundation[30]

Table 2: *Open source licence compatibility with GPLv3+/MPLv2+*

E.6 veraPDF licence compatibility

Given that open source software often reuses multiple other projects, dependency analysis can very quickly uncover a wide variety of licenses in the dependency of any one project. In most cases, these licenses will be compatible, enabling the software to be distributed under one license.

The tables in this section list the proposed dependencies of each veraPDF Conformance Checker component and describe the licence and compatibility with GPLv3+/MPLv2+ based on the legal analysis.

veraPDF dependencies are derived using:

- software identified in the Technical Specification;
- a Maven plugin-generated license and dependency analysis[31] of the veraPDF proof-of-concept (see Annex F: Software and Demonstrator).

[27] http://www.gnu.org/licenses/license-list.html#GPLv2
[28] http://www.gnu.org/licenses/license-list.html#MPL-2.0
[29] http://www.gnu.org/licenses/license-list.html#AGPLv3.0
[30] http://www.gnu.org/licenses/license-list.html#MPL-2.0

[31] http://projects.opf-labs.org/verapdf/pdf-rest/pdfbox-rest-application/dependencies.html

During the first half of Phase 2 we will use the PDFBox PDF Parser however this will be swapped-out with our greenfields PDF Parser when it is available during the redesign stage.

The Apache Software Foundation identifies PDFBox dependencies using:

- a manually maintained list[32];
- a Sonar analysis of the PDFBox code base[33].

We also supply estimated effort, and by implication cost, for redeveloping these components if necessary. This is based on the COCOMO model[34] for estimating development effort. A conservative method for using these figures to calculate the impact on veraPDF development time would be to use a rule of thumb and divide by 5 to estimate the proportion of each project we would have to redevelop.

An example of COCOMO model estimation for JUnit: https://www.openhub.net/p/junit

** figures for some projects were not available on http://openhub.net/*

[32] https://pdfbox.apache.org/dependencies.html

[33] https://analysis.apache.org/plugins/resource/58986?page=org.sonar.plugins.design.ui.libraries.LibrariesPage

[34] http://en.wikipedia.org/wiki/COCOMO

E.6.1 Implementation Checker dependencies

NOTE: these only apply during the first half of Phase 2. Once we have developed the veraPDF greenfield PDF Parser the Implementation Checker will have no dependencies.

Function	Dependency	Licence	Proposed use	Legal justification	Estimated effort/cost
PDF parsing	PDFBox	Apache 2.0[35]	Patching Linking	1 1	30 years
	(required by PDFBox) Preflight XMPBox	Apache 2.0[36]	Patching Linking	1 1	[unknown]*
	(required by PDFBox) FontBox Jempbox	BSD[37, 38]	Patching Linking	3 3	2 years
	(required by PDFBox) Commons Logging	Apache 2.0[39]	Linking	1	3 years
Encryption	(required by PDFBox) Bouncy Castle	MIT[40]	Linking	4	128 years

E.6.2 Metadata Fixer dependencies

Metadata Fixer dependencies are identical to the Implementation Checker - the final prototype will have no dependencies but we propose the first prototype be based on PDFBox to expedite development and ensure that maximum amount of time is available for testing.

[35] https://pdfbox.apache.org/index.html

[36] https://pdfbox.apache.org/index.html

[37] http://sourceforge.net/projects/fontbox/

[38] http://sourceforge.net/projects/jempbox/

[39] http://commons.apache.org

[40] http://www.bouncycastle.org/licence.html

E.6.3 Policy Checker dependencies

Function	Dependency	Licence	Proposed use	Legal justification	Estimated effort/cost
Policy Profiles	Schematron	OSI compliant zlib/libpng license and Apache License[41]	Linking	1	147 years
Policy Checking	ph-schematron	Apache 2.0[42]	Linking	1	55 years
	Probatron4j	Affero GPL v3[43]	Linking	8	2 years

E.6.4 Reporter dependencies

Function	**Dependency**	**Licence**	**Proposed use**	**Legal justification**	**Estimated effort/cost**
PDF Reporting	FOP	Apache 2.0[44]	Linking	1	90 years
XML Reporting	Xalan	Apache 2.0[45]	Linking	1	53 years
Internationalisation	TMX format	CC-BY 3.0[46]	[n/a]	[n/a]	[n/a]
	TMX Validator	Eclipse Public Licence[47]	Linking	6	[unknown]*
	Heartsome TMX Editor	GPL v2[48]	Standalone	7	[unknown]*

[41] http://www.schematron.com

[42] https://github.com/phax/ph-schematron

[43] http://www.probatron.org/probatron4j.html

[44] http://xmlgraphics.apache.org/fop/license.html

[45] http://xalan.apache.org

[46] http://www.gala-global.org/oscarStandards/tmx/tmx14b.html

[47] http://sourceforge.net/projects/tmxvalidator/

[48] https://github.com/heartsome/tmxeditor8/blob/master/LICENSE

E.6.5 Shell dependencies

Function	Dependency	Licence	Proposed use	Legal justification	Estimated effort/cost
REST service framework	JavaX.RS	[within Java]	Linking	[n/a]	[n/a]
Parsing command line parameters	Commons - CLI	Apache 2.0[49]	Linking	1	2 years
Web Interface layout	JQuery	MIT[50]	Linking	4	16 years
	Bootstrap	MIT[51]	Linking	4	17 years
REST application framework and server	DropWizard Core	Apache 2.0[52]	Linking	2	13 years
Monitoring or managing REST web services	(required by DropWizard core) dropwizard-* metrics-*	Apache 2.0[53]	Linking	1	5 years
Java object to JSON serialisation for web services	(required by DropWizard core) jackson-*	Apache 2.0 and LGPL 2.1[54]	Linking	1, 2	23 years
Java web server and servlet container.	(required by DropWizard core) jetty-*	Apache 2.0 and Eclipse Public License 1.0[55]	Linking	1	201 years
Developer support libraries	(required by DropWizard core) findbugs java	LGPL v3[56]	Linking	2	73 years

[49] http://commons.apache.org/proper/commons-cli/
[50] https://jquery.org/license/
[51] http://getbootstrap.com/getting-started/#license-faqs
[52] http://dropwizard.io/about/faq.html
[53] https://github.com/dropwizard/metrics/blob/master/LICENSE
[54] http://wiki.fasterxml.com/JacksonLicensing
[55] http://eclipse.org/jetty/licenses.php
[56] http://findbugs.sourceforge.net

Function	Dependency	Licence	Proposed use	Legal justification	Estimated effort/cost
Java object to XML serialisation for web services	Jackson XML Dataformatter	Apache 2.0 and LGPL 2.1[57]	Linking	1, 2	2 years

E.6.6 High-level dependencies

Function	Dependency	Licence	Proposed use	Legal justification	Estimated effort/cost
Runtime	Linux	[various]	Runtime	[n/a]	[out of scope]
	Java Virtual Machine (HotSpot)	GPLv2	Runtime	[n/a]	[out of scope]
	Java Virtual Machine (IcedTea)	GPL with linking exception	Runtime	[n/a]	[out of scope]
Developing or executing unit tests	JUnit	Eclipse Public License 1.0[58]	Testing	6	7 years
Testing hashCode and equals methods for Java objects	EqualsVerifier	Apache 2.0[59]	Testing	1	3 years
Collections, caching, primitives support, concurrency libraries, common annotations, string processing, I/O, etc.	Guava Libraries	Apache 2.0[60]	Linking	1	71 years

[57] http://wiki.fasterxml.com/JacksonLicensing

[58] http://junit.org/license.html

[59] http://www.jqno.nl/equalsverifier/

[60] https://code.google.com/p/guava-libraries/

Function	Dependency	Licence	Proposed use	Legal justification	Estimated effort/cost
Standard encoding and decoding routines (e.g. Base64 encoding)	Commons - Codec	Apache 2.0[61]	Linking	1	5 years

[61] http://commons.apache.org

Annex F: Software and Demonstrator

veraPDF GitHub account: https://github.com/verapdf/

Name	Web-based demonstrator (pdfbox-rest)	PDFBox fork (for testing)
Description	Online demonstrator of PDF/A parsing and validation functionality (based on PDFBox)	For evaluating PDFBox against functional and technical specifications
URL	http://preflight.verapdf.org/	Not available
Source code	https://github.com/verapdf/pdfbox-rest	https://github.com/verapdf/pdfbox
Jenkins	http://jenkins.opf-labs.org/job/pdfbox-rest/	http://jenkins.opf-labs.org/job/PDFBox/
Sonar	http://sonar.opf-labs.org/dashboard/index?id=org.verapdf.pdfbox%3Apdfbox-rest	http://sonar.opf-labs.org/dashboard/index?did=1&id=org.apache.pdfbox%3Apdfbox-reactor
Travis	https://travis-ci.org/verapdf/pdfbox-rest	https://travis-ci.org/verapdf/pdfbox

Annex G. ICC Profile Checks for PDF/A Validation

G.1 Normative references

[1] PDF/A specifications: ISO 19005-1:2005, ISO 19005-2:2011, ISO 19005-3:2012.
[2] PDF Specifications: PDF 1.4 Specification (Adobe), ISO 32000-1:2008.
[3] ICC specifications: ICC.1:2004-10 (Profile version 4.2.0.0)

G.2 Terminology

Term	Definition
Profile header	The first 128 bytes of the ICC profile as defined by 7.2 of ICC.1:2004-10.
Profile version	The version of the ICC profile as specified in bytes 8-11 of the profile header, see 7.2.4 of ICC.1:2004-10.
Device class	The profile device class as specified in bytes 12-15 of the profile header, see 7.2.5 of ICC.1:2004-10.
Profile colour space	The data colour space as specified in bytes 16-19 of the profile header, see 7.2.6 of ICC.1:2004-10.
Profile connection space (PCS)	Profile connection space as specified in bytes 20-23 of the profile header, see 7.2.7 of ICC.1:2004-10.
Profile tag	A named byte range of the ICC profile defined in the profile Tag Table, see 7.3 of ICC.1:2004-10.
Input profile	An ICC profile with device class “scnr”.

Term	Definition
Display profile	An ICC profile with device class “mntr”.
Output profile	An ICC profile with device class “prtr”.
ColorSpace conversion profile	An ICC profile with device class “spac”.

G.3 ICC profile requirements

G.3.1 Version information

ICC profile version shall agree with the PDF file version as specified in Table 67 of ISO 32000-1:2008.

G.3.2 Device class

Device class of the ICC profile referred by **DestOutputProfile** in the PDF/A **OutputIntent** shall be “prtr” or “mntr”.

Device class of the ICC profile used in the ICCBased colour space array shall be “scnr”, “mntr”, “prtr” or “spac”.

G.3.3 Colour space

Colour space of the ICC profile referred by **DestOutputProfile** in the PDF/A **OutputIntent** shall have a colour space of either “GRAY”, “RGB”, or “CMYK”.

Colour space of the ICC profile referred by **DestOutputProfile** in the PDF/A **OutputIntent** shall have a colour space of either “GRAY”, “RGB ”, “CMYK”, or “Lab ”.

G.3.4 Requirements for specific profile types

G.3.4.1 Profile types

Each profile has one of three types:

- N-component LUT-based
- Three-component matrix-based
- Monochrome

characterized by a different model for conversion between the device colour space and the profile connection space.

There is no explicit type information in the profile header. However, the type can be derived based on the collection of tags present in the profile:

- Presence of the tag “grayTRCTag” implies Monochrome type.
- Presence of one of the following tags “redTRCTag”, “greenTRCTag”, “blueTRCTag” implies Three-component matrix-based profiles.
- If none of these tags is present this implies N-component LUT-based type.

G.3.4.2 Input profiles

Input profile shall have the type of either “N-component LUT-based”, “Three-component matrix-based”, or “Monochrome”.

G.3.4.3 Display profiles

Display profile shall have the type of either “N-component LUT-based”, “Three-component matrix-based”, or “Monochrome”.

G.3.4.4 Output profiles

Output profile shall have the type of either “N-component LUT-based” or “Monochrome”.

G.3.4.5 Profile connection space

Any Three-component matrix-based profile shall use “XYZ ” as a profile connection space.

G.3.4.6 Required tags per each profile type

The following tags are required for all profile types:

- profileDescriptionTag
- mediaWhitePointTag
- copyrightTag
- chromaticAdaptationTag

The additional required tags per profile type are:

Profile type	Required tags	Additional permitted tags
N-component LUT-based input profile	AToB0Tag	AToB1Tag AToB2Tag BToA0Tag BToA1Tag BToA2Tag gamutTag
N-component LUT-based display profile ColorSpace conversion profile	AToB0Tag BToA0Tag	AToB1Tag AToB2Tag BToA1Tag BToA2Tag gamutTag

Profile type	Required tags	Additional permitted tags
N-component LUT based output profile	AToB0Tag BToA0Tag gamutTag AToB1Tag BToA1Tag AToB2Tag BToA2Tag	
Monochrome input profile Monochrome display profile Monochrome output profile	grayTRCTag	AToB0Tag AToB1Tag AToB2Tag BToA0Tag BToA1Tag BToA2Tag
Three-component matrix-based input profile Three-component matrix-based display profile	redMatrixColumnTag greenMatrixColumnTag blueMatrixColumnTag redTRCTag greenTRCTag blueTRCTag	AToB0Tag AToB1Tag AToB2Tag BToA0Tag BToA1Tag BToA2Tag gamutTag

G.3.5 Tag definitions

All tags used in the ICC profile shall have one of the permitted tag types as specified in Section 9 of ICC.1:2004-10 and the corresponding type definitions as specified in Section 10 of ICC.1:2004-10.

The number of input and output channels for tags defining colour transform shall agree with the values of both the profile colour space and the profile connection space.

Annex H. Embedded Font Checks for PDF/A Validation

H.1 Normative references

PDF/A specifications: ISO 19005-1:2005, ISO 19005-2:2011, ISO 19005-3:2012.

PDF Specifications: PDF 1.4 Specification (Adobe), ISO 32000-1:2008.

Font specifications:

[1] Apple Computer, Inc., TrueType Reference Manual. Available on Apple's Web site at http://developer.apple.com/fonts/TTRefMan/

[2] Microsoft Corporation, TrueType 1.0 Font Files Technical Specification. Available at http://www.microsoft.com/typography/tt/tt.htm

[3] Microsoft Corporation, OpenType specification, version 1.6. Available at http://www.microsoft.com/typography/otspec/

[4] Open Font Format, ISO/IEC 14496-22:2009 (Second Edition).

[5] Adobe Type 1 Font Format, Adobe Systems Incorporated, ISBN 0-201-57044-0, 1990.

[6] Technical Note #5015, Type 1 Font Format Supplement, 15 January 1994, Adobe Systems Incorporated.

[7] Technical Note #5088, Font Naming Issues, 12 April 1993, Adobe Systems Incorporated.

[8] Technical Note #5092, CID-Keyed Font Technology Overview, Adobe Developer Support, 12 September 1994, Adobe Systems Incorporated.

[9] Technical Note #5176, The Compact Font Format Specification, Version 1.0, 18 March 1998, Adobe Systems Incorporated.

[10] Technical Note #5177, The Type 2 Charstring Format, 5 May 1998, Adobe Systems Incorporated.

[11] Technical Note #5641, Enabling PDF Font Embedding for CID-Keyed Fonts, 7 July 1998, Adobe Systems Incorporated.

[12] PostScript Language Reference, Third Edition, Adobe Systems Incorporated, ISBN 0-201-37922-8, 1999.

H.2 Terminology

Term	Definition
CFF Charset	A structure in the CFF Font providing the mapping from the glyph names to GIDs. See [9], Section 13.
CFF Font	Type1 or CIDFontType0 font file in compact font format as specified by [9, 10].
CharStrings Dictionary	A dictionary associating character names (the keys in the dictionary) with glyph descriptions in PostScript Type1 Font file. See [5], Chapter 6.
CharStrings INDEX	An array of all glyph descriptions in CFF Font. See [9], Section 14.
GID	An index used to identify glyph description either in the Charstings INDEX of the CFF Font or in “glyf” table of the TrueType or OpenType Font.
OpenType Font	OpenType font file as specified by technically equivalent documents [3, 4].
PDF Font Descriptor Dictionary	A font descriptor dictionary referred by **FontDescriptor** key in the PDF Font Dictionary and specified in ISO 32000-1:2008, 9.8.
PDF Font Dictionary	Either a simple font dictionary as specified by ISO 32000-1:2008, 9.6 or a CIDFont dictionary as specified in ISO 32000-1:2008, 9.7.4.
PDF Font File Stream	A PDF stream containing the embedded font program (file) referred by one of the keys **FontFile**, **FontFile2**, **FontFile3** in the PDF Font Descriptor Dictionary and specified in ISO 32000-1:2008, 9.9.
PDF Font Type	The value of the key **Subtype** in the PDF Font Dictionary.
PostScript Type1 Font	Type1 font file in PostScript format as specified by [5].
TrueType Font	TrueType font file as specified by technically equivalent documents [1, 2].
TrueType/OpenType table	A named byte range of the TrueType or OpenType font file as defined in [1-4].

H.3 Formats of embedded font files

PDF specification supports the following formats of embedded font files:

PDF Font Type	Key in the PDF Font Descriptor Dictionary	Value of Subtype key in the PDF Font File Stream	Font file format	Normative reference
Type1, MMType1	FontFile	-	PostScript Type1	[5, 12]
Type1, MMType1	FontFile3	Type1C	CFF	[9]
Type1	FontFile3	OpenType	OpenType with “CFF” table	[3,4,9]
TrueType	FontFile2	-	TrueType	[1,2]
TrueType	FontFile3	OpenType	OpenType with “glyf” table	[3,4]
CIDFontType0	FontFile3	CIDFontType0C	CFF	[9]
CIDFontType0	FontFile3	OpenType	OpenType with “CFF” table	[3,4,9]
CIDFontType2	FontFile2	-	TrueType	[1,2]
CIDFontType2	FontFile3	OpenType	OpenType with “glyph” table	[3,4]

H.4 Embedded font file requirements

H.4.1 PostScript Type1 Fonts

The values of keys **Length1**, **Length2**, **Length3** of the PDF Font File Stream shall be correct.

The general font file organization shall comply to [5], Chapter 2.

The glyph with name “.notdef” shall be present in the CharStrings dictionary.

The font file dictionary shall contain a valid Encoding array as specified by [5], 2.2 and [12], 5.3.

Names of all glyphs referenced for rendering shall be present in the CharStrings dictionary. A glyph name is referenced for rendering if it is mapped from the character referenced for rendering via the Encoding mechanism for Type1 fonts as specified by ISO-32000:1, 9.6.6.2.

If the **CharSet** key is present in the PDF Font Descriptor Dictionary, the names of all glyphs specified in its value shall be present in the font CharStrings dictionary, regardless of whether this glyph is referenced for rendering or not.

Charstrings for all glyphs in 4.1.2, 4.1.4, 4.1.5 shall comply with the charstring encoding specification in [5], Chapter 6.

Glyph widths referenced for rendering shall be consistent with the width information in PDF Font Dictionary. Glyph widths in the PostScript Type1 file are determined by the Metrics dictionary of the font file (see [5], 2.2; [12], 5.9.2) or, if it is not present, by “hsbw” or “sbw” operator in the glyph charstring (see [5], 6.4).

H.4.2 Compact Font File (CFF) in case of Type1 and MMType1 PDF Font Types

The general CFF Font file structure shall comply to [9], Section 2 and shall consist only of a single font. In particular, it shall contain a valid Header, Name INDEX, Top DICT INDEX, String INDEX, Global Subr INDEX, Encoding, Charset, CharStrings INDEX, Font DICT INDEX, Private DICT.

All GIDs referenced for rendering from CIDs via the algorithm defined in ISO 32000-1:2008, 9.7.4.2 shall be present in the CharStrings INDEX.

If the **CharSet** key is present in the PDF Font Descriptor Dictionary, the names of all glyphs specified in its value shall be present in the Charset structure, regardless of whether this glyph is referenced for rendering or not.

GIDs for all glyphs in 4.2.2, 4.2.3 identified via Charset structure, shall point to valid charstings in the CharStrings INDEX as specified by [9], Section 14; [10].

Glyph widths referenced for rendering shall be consistent with the width information in PDF Font Dictionary. Glyph widths in the CFF file are determined by “hsbw” or “sbw” operator in the glyph charstring of Type1, or as a first number of the Type2 charstring serving as a difference to *nominalWidthX*, or, if omitted, as a *defaultWidthX* (see [10], 3.1). The *nominalWidthX* and *defualtWidthX* are defined in the Private DICT of the CFF font.

H.4.3 Compact Font File (CFF) in case of CIDFontType0 PDF Font Type

The general CFF Font file structure shall comply to [9], Section 2 and shall consist only of a single font. In particular, it shall contain a valid Header, Name INDEX, Top DICT INDEX, String INDEX, Global Subr INDEX, CharStrings INDEX, Font DICT INDEX, Private DICT and, optionally, the FDSelect structure. Both the Encoding and the Charset structures are optional are not used for locating glyph charstings.

All GIDs mapped from CIDs used for rendering via an algorithm defined in 9.7.4.2 shall be present in the font file and correctly encoded.

Glyph widths referenced for rendering shall be consistent with the width information in PDF Font Dictionary. Glyph widths in the CFF file are determined by “hsbw” or “sbw” operator in the glyph charstring of Type1, or as a first number of the Type2 charstring serving as a difference to nominalWidthX, or, if omitted, as a defaultWidthX (see [10], 3.1). The nominalWidthX and defualtWidthX are defined in the Private DICT of the CFF font.

H.4.4 TrueType Font File in case of TrueType PDF Font Type

The font shall contain the following minimal set of tables: “cmap”, “glyf”, “head”, “hhea”, “hmtx”, “loca”, “maxp”. The “cvt ”, “fpgm”, and “prep” tables must also be included if they are required by the font instructions. All these tables shall comply to the data format requirements of [1,2].

If the PDF Font Descriptor **Flags** key identifies the PDF Font as non-symbolic (ISO 32000-1:2008, 9.8.2) and the PDF Font Encoding defines Differences array, then the “cmap” table shall contain at least Microsoft Unicode (3,1 – Platform ID=3, Encoding ID=1) encoding.

In case of PDF/A-1 standard, if the PDF Font Descriptor Flags key identifies the PDF Font as symbolic (ISO 32000-1:2008, 9.8.2), then the “cmap” table shall contain exactly one encoding.

In case of either PDF/A-2 or PDF/A-3 standard, if the PDF Font Descriptor Flags key identifies the PDF Font as symbolic (ISO 32000-1:2008, 9.8.2), then the “cmap” table shall either contain exactly one encoding or at least Microsoft Symbol (3,0 – Platform ID=3, Encoding ID=0) encoding. If Microsoft Symbol encoding is present, the range of character codes shall be one of these: 0x0000 - 0x00FF, 0xF000 - 0xF0FF, 0xF100 - 0xF1FF, or 0xF200 - 0xF2FF.

All GIDs of glyphs used for rendering, as determined by the algorithm described in ISO 32000-1, 9.6.6.4, shall be present in the “glyf” table and their instructions shall comply with [1,2].

All GIDs of glyphs used for rendering, as determined by the algorithm described in ISO 32000-1, 9.6.6.4, shall be present in the “hmtx” table and their widths shall be consistent with widths information of the PDF Font Dictionary.

H.4.5 TrueType Font File in case of CIDFontType2 Font Type

The font shall contain the following minimal set of tables: “glyf”, “head”, “hhea”, “hmtx”, “loca”, “maxp”. The “cvt ”, “fpgm”, and “prep” tables must also be included if they are required by the font instructions. The tables “vhea” and “vmtx” shall also be included if the PDF Font is used for vertical writing. All these tables shall comply to the data format requirements of [1,2].

All GIDs of glyphs used for rendering, as determined by the algorithm described in ISO 32000-1, 9.7.4.2, shall be present in the “glyf” table and their instructions shall comply with [1,2].

All GIDs of glyphs used for rendering, as determined by the algorithm described in ISO 32000-1, 9.7.4.2, shall be present in the “hmtx” table and their widths shall be consistent with widths information of the PDF Font Dictionary (keys **W** and **DW**). If the PDF Font is used for vertical writing, the same condition applies to “vmtx” table and metrics information of the PDF Font Dictionary (keys **W2** and **DW2**).

H.4.6 OpenType fonts

An OpenType font file shall not contain both “glyf” and “CFF ” tables.

If an OpenType font file contains “glyf” table shall comply either with the requirements of Section 4.4 in case of TrueType PDF Fonts or with the requirements of Section 4.5 in case of CIDFontType2 PDF Fonts.

If an OpenType font file contains “CFF ” table, its data shall comply either with the requirements of Annex G, 4.2 in case of Type1 or MMType1 PDF Fonts or with the requirements of Annex G 4.3 in case of CIDFontType0 PDF Fonts.