veraPDF Policy Checking

Carl Wilson, Open Preservation Foundation
Boris Doubrov, Dual Lab

Industry-supported
PDF/A validation

Validation & Policy : PDF/A Validation

PDF/A validation is veraPDF’s raison d'etre, but validation is not always enough:

- Somebody else gets to make the rules.
- The assumption is that “one size fits all”, or eight sizes for PDF/A standards.
- The PDF/A standards are published infrequently.
- The standards are not easy to change, nor should they be.
- Beyond disallowing some of them, the PDF/A standards don’t make many assertions about formats embedded within PDF documents, e.g. images, fonts.

Validation & Policy : Policy Restrictions

Institutions may wish to enforce different criteria for the PDF/A documents that they preserve. These criteria will often make demands beyond the PDF/A specifications, for example:

- Disallowing particular image compression types.
- Restricting the set of fonts used in documents.
- Quality assuring metadata embedded in PDFs.

Validation & Policy : Relaxing Validation Criteria

Alternatively, policy can be used to relax PDF/A validation criteria, examples might be:

- Allowing some or any external fonts.
- Accepting forms of compression disallowed by PDF/A.

To be clear, these aren’t policy recommendations, they’re just examples. PDFs with these properties aren’t valid PDF/As but pragmatism might have to prevail. One case might be that there is a known problem with a set of PDFs and the collecting institution can’t change them.

Feature Reporting : What is a PDF Feature

- Any information on the PDF document and its internals that might be relevant for characterization and automatic processing of the document
- Parsed from PDF syntax and converted to machine-readable XML syntax

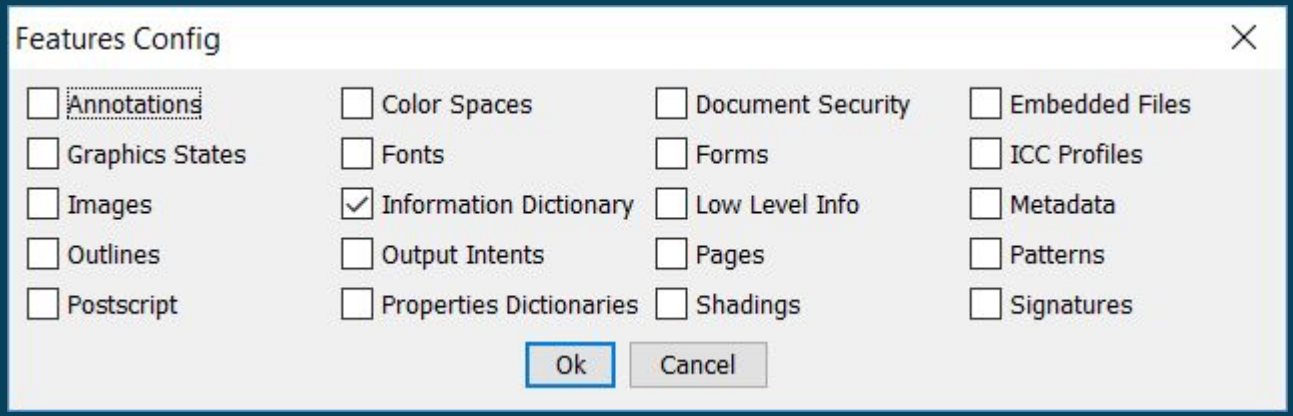
  Examples:
  - Number of objects in the PDF document
  - Font names
  - Color models

Feature Reporting : Handle with Care

- Extracting all features might be dangerous!
- Normally would result in the Feature report of size comparable with the original PDF (so, 100 Mb would not be unusual)
- PDF document may contain millions of features (we do have real world examples like this)
- Would slow down the validation process considerably
- Requires a lot of memory
- **Advice: extract only the features you need!**

Feature Reporting : Configuration (GUI)

- File->Features Config



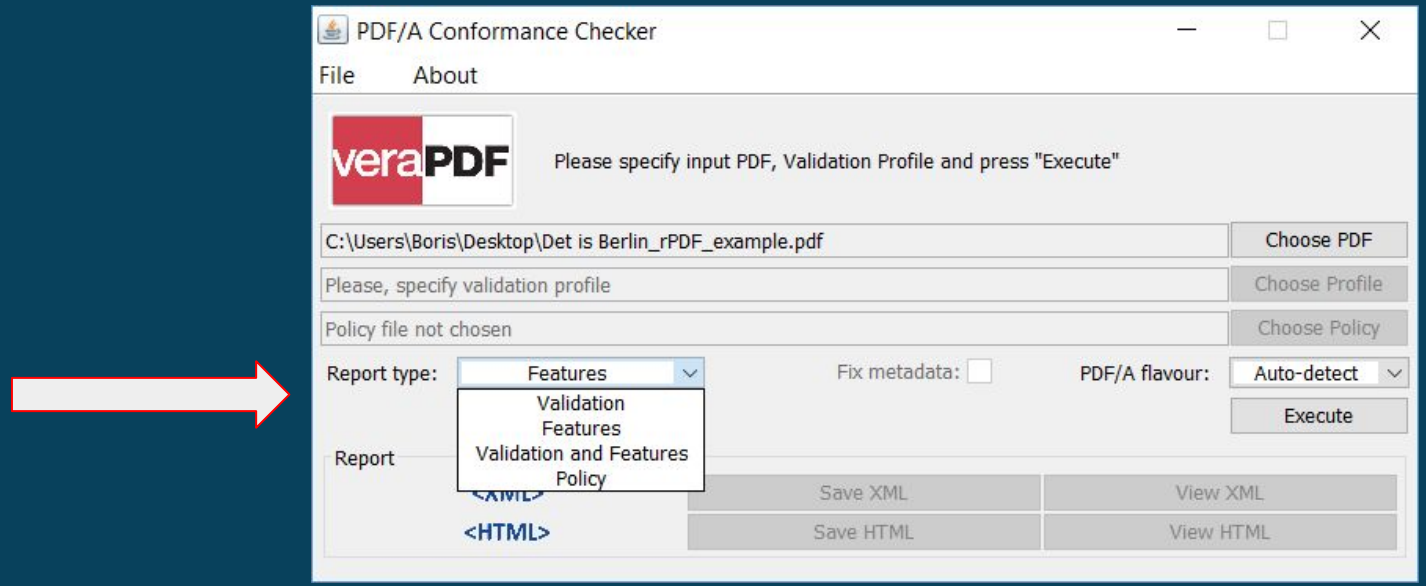
Feature Reporting : Configuration (CLI and API)

- <installation folder>/config/features.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<featuresConfig>
    <enabledFeatures>
        <feature>INFORMATION_DICTIONARY</feature>
    </enabledFeatures>
</featuresConfig>
```

Feature Reporting : Inspecting a Report

- Experiment: Generate the feature report and open it in any XML viewer



Feature Reporting : Inspecting a Report

```
<featuresReport>
    <informationDict>
        <entry key="Title">Det is Berlin - wie wunderbar!</entry>
        <entry key="Creator">Adobe InDesign CC 2017 (Macintosh)</entry>
        <entry key="Producer">Adobe PDF Library 15.0</entry>
        <entry key="CreationDate">2017-02-08T16:43:00.000+01:00</entry>
        <entry key="ModDate">2017-03-29T10:30:46.000+03:00</entry>
        <entry key="Trapped">False</entry>
    </informationDict>
</featuresReport>
```

Documentation on all features

- Features are split into 20 categories
- Categories are visible in the GUI Features config dialog
- More details on the features at: http://docs.verapdf.org/cli/config/#features.xml
- Full list:
  - Annotations, Color Spaces, Document Security, Embedded Files, Graphics States, Fonts, Forms, ICC profiles, Images, Information Dictionary, Low Level Info, Metadata, Outlines, Output Intents, Pages, Patterns, PostScript, Property Dictionaries, Shadings, Digital Signatures

If you need more features

- The veraPDF plug-in framework allows to inject additional features into the Feature Report
- Plug-ins get raw data of:
  - Images
  - Font files
  - ICC profiles
  - Digital certificates
  - XMP Packages
  - Attached files
- All data except for Images is uncompressed. So, no PDF-specific knowledge is needed to inspect them in more detail

Demonstration…..

XML Schematron : An Introduction

The veraPDF policy checker is implemented as an XML Schematron engine. Schematron is an XML syntax that :

- Allows the user to express constraints, known as assertions, about XML data.
- Is designed for quality assurance, expressing business rules and XML validation.
- Is an ISO standard with a variety of open source implementations available.
- Is a simple syntax that requires only five basic elements to be useful.

XML Schematron : Supporting Technologies

Underpinning Schematron are two supporting XML technologies:

- XPath : used to define the elements of interest in an XML document.
- XQuery : used to write queries against XML data.

Anyone comfortable with these or similar technologies, for example XSLT, should find many familiar concepts when starting out with Schematron.

We'll show a little of these as we go but only as much as is absolutely necessary.

XML Schematron : An Overview

The following example, taken from the Schematron site, shows a single rule that checks an HTML document for a date in a Dublin Core metadata field:

```
<rule context="/html/head">
  <assert test=
  "string-length(meta[@name="dc.date"]/@CONTENT) &gt; 9">
  Our HTML documents need a Dublin Core metadata DATE field,
  because of our corporate document retention policy.
  </assert>
</rule>
```

The rule's context attribute is our first XPath example and selects children of the HTML header, while the assert's test attribute is an XPath function checking for a meta tag named dc.date with content that's a string longer than 10 characters.

This IS the content you are looking for

```
<meta name="dc.date" CONTENT="2017-03-30">
```

Unfortunately so is this

```
<meta name="dc.date" CONTENT="onceuponatime">
```

and this

```
<meta name="dc.date" CONTENT="3000-17-35">
```

Policy Checking : Workflow

- The Policy Checker executes Schematron checks on the veraPDF Feature Report
- It’s capable of parsing and compiling Schematron syntax as well as recognising pre-compiled Schematron XSLT documents
- Feature Extraction is a prerequisite of Policy Checking

Policy Checking : A Simple Example

The best way to become familiar with schematron is to study worked examples. Below we show a simple veraPDF policy statement that uses four of the basic schematron elements:

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt">
  <sch:pattern name="Information Dictionary Checks">
    <sch:rule context="featuresReport/informationDict">
      <sch:assert test="count(entry[@key='Title']) > 0">No title.</sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Let’s look a little closer…..

Policy Checking : schema and patterns

The first line sets up schematron namespace and xslt version

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt">
```

The second line is the pattern element which has a name, this can be used to group rules

```
<sch:pattern name="Information Dictionary Checks">
```

A pattern contains a series of rules, any node in the feature report will be checked against the first rule it matches only. So far so good, let’s have a quick look back at a feature report...

Feature Report : A second look

```
<featuresReport>
    <informationDict>
        <entry key="Title">Det is Berlin - wie wunderbar!</entry>
        <entry key="Creator">Adobe InDesign CC 2017 (Macintosh)</entry>
        <entry key="Producer">Adobe PDF Library 15.0</entry>
        <entry key="CreationDate">2017-02-08T16:43:00.000+01:00</entry>
        <entry key="ModDate">2017-03-29T10:30:46.000+03:00</entry>
        <entry key="Trapped">False</entry>
    </informationDict>
</featuresReport>
```

Policy Checking : rules and assertions

Here’s an xPath example that selects the children of the the “informationDict”

```
<sch:rule context="featuresReport/informationDict">
```

Now the assertion counts the number of entries with the key “Title” and makes sure there’s more than none

```
<sch:assert test="count(entry[@key='Title']) > 0">No title.</sch:assert>
```

This IS the content you are looking

```
<entry key="Title">Det is Berlin</entry>
```

Unfortunately so is this

```
<entry key="Title"></entry>
```

Although, null COULD be a legitimate title perhaps?

What about spaces?

And what happens if there are two title entries??

Demonstration…..

Policy Checking : Get involved

Here’s how you could get started with feature reporting and policy checking:

- Give it a go, there’s a few examples online at: http://docs.verapdf.org/policy/
- Tell us how we can improve the software
- Join the mailing list and ask questions

Policy Checker : Development Plans

The veraPDF policy checker is quite flexible and powerful, thanks in most part to Schematron. There’s certainly room for improvement, over the second quarter of 2017 we’ll be:

- Improving feature reporting to provide better support for policy checking.
- Working on a GUI for users without Schematron expertise.
- Working on HTML reporting of features and policy issues.
- Gathering feedback from our users to inform future plans.
- Practising our “Schematron Fu” and improving the policy checker as we learn.

Questions?

- http://verapdf.org/
- users@lists.verapdf.org (Q&A, discussions)
- https://github.com/veraPDF
- http://verapdf.org/subscribe/ (news alerts)
- @_verapdf
- info@verapdf.org

veraPDF

Industry-supported
PDF/A validation

PREFORMA