5 April 2021.

Work in progress: Demystifying PDF through a machine-readable definition

Peter Wyatt[1]

[1] PDF Association Inc., MA 01890 USA

Corresponding author: Peter Wyatt (e-mail: peter.wyatt@pdfa.org).

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001119C0079. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA). Approved for public release.

ABSTRACT This paper presents the Arlington PDF Model as the first open access, comprehensive specification-derived machine-readable definition of all formally defined PDF objects and data integrity relationships. This represents the bulk of the latest 1,000-page ISO PDF 2.0 specification and is a definition for the entire PDF document object model, establishing a state of the art "ground truth" for all future PDF research efforts and implementers. Expressed as a set of text-based TSV files with 12 data fields, the Arlington PDF Model currently defines 514 different PDF objects with 3,551 keys and array elements and uses 40 custom predicates to encode over 5,000 rules. The Arlington PDF Model has been successfully validated against alternate models, as well as a sizeable corpus of extant data files and has been widely shared within the SafeDocs research community as well as the PDF Association's PDF Technical Working Group. It has already highlighted various extant data malformations and triggered multiple changes to the PDF 2.0 specification to reflect the de-facto specification, remove ambiguities, and correct errors.

INDEX TERMS PDF, Portable Document Format, machine-readable definition, domain-specific languages, DSL, data definition languages, DDL, parsing, document object model, DOM

I. Introduction

The Portable Document Format (PDF) is the most ubiquitous electronic document format in use today. Recent media[1] estimates that there are more than 2.5 trillion PDF files in existence, with a major vendor reporting that 303 billion PDFs were opened by just their products in the 2020 fiscal year alone (a 17% annual increase). These are staggering numbers considering that it has been almost 30 years since John Warnock, the co-founder of Adobe, wrote his original Camelot Project whitepaper ([2]), outlining the goals of PDF.

With this ubiquity comes expectations of reliability, robustness, and security. Over the years, many researchers across academia, private enterprise and government agencies have discovered numerous vulnerabilities in PDF implementations, information leakage ([28],[29]), data obfuscation ([30],[31]) and inconsistencies in displaying and processing PDFs ([26], [27]). Much of this work has focused on analyzing and experimenting with PDF implementations, and only little attention has been given to a formal machine-readable definition of PDF ([24],[25]).

The PDF file format is defined by a large specification document and was most recently published as ISO 32000-2:2020 ([5]). PDF is a binary container file format that depends on many other formats and data specifications. The PDF 2.0 standard itself is 1,000 pages with 79 direct normative references[2]. This is an extraordinarily complex document for developers that can easily result in misunderstandings, incorrect implementations, interoperability issues, and inevitably vulnerabilities.

This paper presents the "Arlington PDF Model" ([1]) which is a novel machine-readable definition of the PDF 2.0 document object model (DOM). It provides a structured data definition of every formally defined PDF object and all object and data relationships.

The Arlington PDF Model provides a specification-derived "ground truth" of the full PDF object model which

[1] https://marker.medium.com/the-improbable-tale-of-how-the-lowly-pdf-played-the-longest-game-in-tech-d143d2ba9abf

[2] https://www.pdfa.org/iso-32000-normative-references/

allows a deeper formal analysis of the PDF grammar, rapid identification of extant data deviations from the specification (commonly referred to as the *de-facto specification*) as well as facilitating high quality PDF-based research without the need for deep file format expertise.

In the process of conducting this work we identified and corrected numerous issues in the PDF 2.0 (ISO 32000-2:2017; [4]) standard, ranging from minor inconsistencies to major deficiencies; we identified errors and differences in a popular alternate machine-readable definition of PDF 1.7 as well as identifying previously unreported malformations in extant PDF files which resulted in a change to PDF 2.0 specification to reflect the de-facto specification. The current Arlington PDF Model defines 514 different PDF objects with 3,551 keys and array elements and uses 40 custom predicates to encode over 5,000 rules. We also identify errors in past research and concretely demonstrate how the application of the Arlington PDF Model can improve research outcomes.

This paper begins with a brief overview of different approaches to defining file formats, before examining the 30-year history and evolution of PDF from proprietary specification to an open international standard. This establishes the current environment of the PDF file format and enables a broader understanding of corpora, implementation, and specification divergence. This section concludes with a brief summary of some technical details on the PDF file format and syntactic constructs.

Section III *Related Work* describes the merits and disadvantages of prior work in machine-readable definitions for PDF and how the Arlington PDF Model resolves these issues. This section concludes with concrete examples of how the Arlington PDF Model can be applied across a diverse range of research areas, and how errors in past papers could have been easily identified without the need for deep file format experts.

Section IV *Methodology* describes in detail the goals, process, and evolution of the Arlington PDF Model along with technical details of the model. Section V *Model Validation* then describes how the Arlington PDF Model was successfully validated against another popular model as well as the de-facto specification defined by a large corpus.

The paper concludes with reflections from the creation process, ideas for Future Work and a summary of the many benefits that have already been achieved from the creation of the Arlington PDF Model.

II. Background

A. File format specifications

Traditionally file formats are defined by purely prose-based specifications, developed by one set of experts in a standards environment, and later interpreted by software engineers during implementation. Software engineers tasked with building parsers (readers) or writers for complex formats often do not see all the nuances that are inherent in any large technical specification. Although this re-interpretation process may be supplemented by file format experts (such as experts involved in the standardization process, or those with prior experience); detailed technical examination of extant data files; or detailed technical experimentation of alternate extant parser behaviors (selected because of their market position or source code availability), the implementation outcomes will undoubtedly differ between software engineers. When considered across multiple implementers in a larger ecosystem, this ad-hoc human-centered process results in multiple distinct parsers and malformed files (from error-prone writers) that lead to non-interoperable implementations, parser differentials, and input vulnerabilities.

Some complex page description formats such as Microsoft® XPS[3], PODi PPML 3.0, Adobe® PostScript and HP® PCL have supplemented their specifications with official or de-facto reference implementations, whereby ambiguities in the specification might be resolvable by examining the output or behavior of a specific implementation. In the case of proprietary specifications, the reference implementation is often the 'black box' implementation of the primary vendor. PDF does not have an official reference implementation.

In contrast, the WHATWG HTML Living Standard[4] exists as a duality between the prose-based specifications and agreement between a set of largely 'white box' implementations. However, website consistency between browsers still leaves a lot to be desired due to the vast number of idiosyncrasies in parsing HTML ([6]).

In a similar manner, 'white box' evaluation and experimentation platforms (i.e., implementations) are also used in conjunction with prose-based specifications when developing various low-level image and video formats[5].

B. Specifications and essential truths

As Owen eloquently states in [22]:

"*Formats are specifications. They are not essential truths about files. You can think of formats as something akin to what a priest reads from the text in a ceremony and the files are like all the various ways that people live their beliefs in their everyday life once they leave a place of worship. …*
Many files are authentically invalid, and their invalidity is something that may itself be significant. All conversations about formats need to start from the understanding that they conventions for how files are supposed be structured, not essential truths about how files are structured."

[3] Global Graphics Annual Report 2007, p.24 and https://www.globalgraphics.com/news/ggpress.nsf/GGRVPressReleasesPublished/74FB6274023AC4268025717600548BC4.

[4] https://whatwg.org/

[5] Including JPEG (https://jpeg.org/), VCEG (https://www.itu.int/en/ITU-T/studygroups/2017-2020/16/Pages/video/vceg.aspx) and VQEG (https://www.its.bldrdoc.gov/vqeg/vqeg-home.aspx).

This concept can be graphically represented with unused areas of a file format specification lacking extant data and parser support, malformations of all kinds extending beyond the specification, and various reader/writer implementations all supporting their own subtly different interpretations of the same format specification (Figure 1, not to scale):

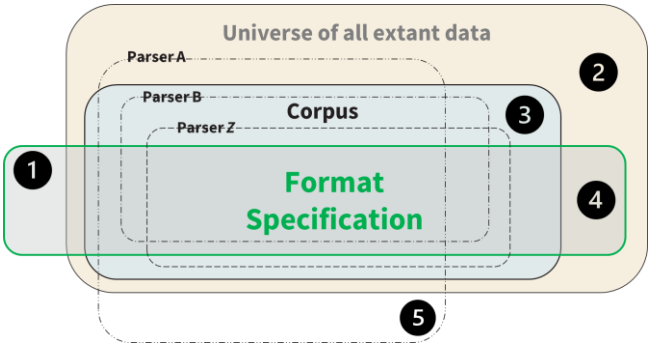

Figure 1: Specifications are not truths about files.

Area ❶ in Figure 1 represents that portion of a specification which is unknowingly unimplemented across all parsers and writers, and thus outside the universe of all extant data. Whenever it can be identified, it is an area ripe for removal from a specification. Area ❷ is the universe of all extant data files, representing files that are both within and outside the formal specification. In the case of PDF this is at least 2.5 trillion files! Area ❸ represents an arbitrary corpus that will undoubtedly miss both samples of extant data malformations and areas of the format specification (as indicated by zone ❹). In the case of PDF, this will likely include entire classes of document that are not readily available via public internet searching (see [32]). Each parser implementation uniquely intersects these areas, either with intentional support for certain malformations or accidental and unintended permissiveness – possibly to the extent of zone ❺, representing an unwitting and unreachable area by any writer. Area ❺ is precisely the kind of "weird machine" zone targeted by malicious threat actors!

When corpus-based and experimentally-based research is considered in light of Figure 1, it is apparent that results depend on the zones covered and extrapolation of results may not always be valid. Furthermore, constant changing of file reader and file writer software can alter reproducibility.

What is needed is a stable "ground truth" baseline from which all research can be built. The Arlington PDF Model forms a machine-readable definition of PDF which establishes a specification-derived "ground truth" from which corpora and extant parsers will differ. This difference is to be expected, but Arlington provides a yardstick which can be used to describe and assess such variations.

C. Evolution of the PDF Specification

The difficulty in creating the Arlington PDF Model (and the need for such a model in the first place) is due to the size, complexity, and subtleties in the PDF specification. Many subtle but important details are easily overlooked in the prose descriptions, and only experts who were actively involved in the standardization process are cognizant. A specification-derived machine-readable model of the official PDF object model can encode such knowledge and ensure that all requirements are extracted and captured.

Authoring of the first PDF 1.0 specification occurred within Adobe Systems between the publication of the seminal "*The Camelot Project*" whitepaper ([1]) by Adobe's Dr John Warnock in 1991 until the PDF 1.0 specification was first published in June 1993. In comparison to the formalities of more modern technical specifications based on specific language constructs defined by the likes of RFC 2119 ([7]) or the ISO Directives Part 2 ([8]), the original PDF 1.0 specification used a less formal factual-style wording to communicate requirements about the format. However, Adobe did include limited formal Backus-Naur Form (BNF) expressions in their PDF 1.0 specification, which continued up to PDF 1.3 (2000) when all BNF was removed by Adobe.

In early 2007, Adobe passed their latest PDF 1.7 specification to ISO to seed the standardization process towards ISO 32000-1:2008 ([3]), which was published in July 2008[6]. This 18-month ISO "fast track" process was primarily to convert the less formal Adobe PDF 1.7 specification wording into formal ISO language and document requirements (as required by [8]). In ISO standards normative requirements use the word "shall", while recommendations use the word "should".

Subsequently ISO 32000-2 (PDF 2.0) was developed between 2008 and 2017 under the full ISO consensus-based process, with many more significant wording changes, including a few sub-clause rewrites, with substantial efforts to improve clarity of targeted technical requirements by meticulously crafted wordsmithing. A significant number of new features were also introduced with PDF 2.0, however a full review and rewrite of the entire PDF specification was never attempted.

As Rosenthol outlines in [9], PDF defines 4 parts in every PDF file: the header, the body, cross-reference, and the trailer. In addition are 8 text-based syntaxes[7], several "native" binary formats, 3 text encodings (`PDFDocEncoding`, UTF16-BE, UTF-8), 10 stream filters, and many other binary formats that can be embedded in PDF files. An indicative measure of the number of external formats that can be embedded in PDF files is the number of Normative References, shown in *Table 1: PDF Timeline and Version Complexity* below. In PDF 2.0, there are 70 normative references which expands to 210 normative references at the 2nd level and 806 at the 3rd level, when all nested normative references are followed.

[6] https://www.adobe.com/devnet/pdf/pdf_reference.html

[7] Rosenthol also lists XFA in [9], but this was officially deprecated in PDF 2.0.

PDF Version	Date	Pages	Page Size	Complexity Measure
Adobe PDF 1.0	June 1993	230	Book	43 tables, 42 figures
Adobe PDF 1.1	January 1996	302	Book	20 references
Adobe PDF 1.2	November 1996	394	Book	137 tables, 86 examples
Adobe PDF 1.3	July 2000	696	Book	223 tables, 73 figures
Adobe PDF 1.4	December 2001	978	Book	277 tables, 20 color plates
Adobe PDF 1.5	August 2003	1172	Book	333 tables, 70 figures
Adobe PDF 1.6	November 2004	1236	Book	370 tables, 80 figures
Adobe PDF 1.7	October 2006	1310	Book	389 tables, 98 figures
ISO 32000-1:2008 (PDF 1.7)	July 2008	756	A4	5474 "shall", 391 "should", 78 normative references
ISO 32000-2:2017 (PDF 2.0)	July 2017	984	A4	5811 "shall", 410 "should", 91 normative references
ISO 32000-2:2020 (PDF 2.0)	December 2020	1002	A4	5891 "shall", 413 "should", 80 normative references

Table 1: PDF Timeline and Version Complexity

Thus, the PDF specification can be summarized as a complex legacy container file format specification, originally written by a single organization in less rigorous language, that has subsequently evolved to use more modern language in some parts. Compounding this are a large number of dependent technical publications (normative references) and foreign language editions[8] where nuances of technical English requirements may have been lost in translation.

A solution to the problem of ambiguities in prose-specifications are machine-readable definitions for a file format, such as schemas, data definition languages (DDLs) and domain specific languages (DSLs). Schemas typically define various constraints on structure and content and can include data integrity and relationship rules. DDLs are highly specialized languages that allow an expert to formalize a file format grammar so that verifiable statements about a grammar can be proven. DSLs are targeted custom languages that can be defined and developed for any specific purpose, including grammar description and validation.

For reading (parsing) and writing a file format sufficiently to safely confirm file validity, a machine-readable definition such as the Arlington PDF Model will be sufficient. However, for something as complex as PDF which also prescribes interactive behaviors, explicit appearance models, and many other non-parser-related requirements, a machine-readable definition of a file format must still be supplemented by a prose-based description.

D. The PDF object model

In its simplest form a PDF file comprises four main parts: the file header, the file body, the cross-reference table, and the trailer; with each part having subtly different lexical rules. The short file header identifies the file as a PDF file. The file body is typically the largest part of a file and consists of a sequence of indirect PDF objects that form the tree-like DOM representing the paginated document. An indirect object is a standalone PDF object using keywords '`obj`' and '`endobj`' that represents a node in the DOM. An indirect PDF object is referenced by a pair of integers known as its object and generation numbers which are the integers that occur before the '`obj`' keyword. Together the cross-reference table and trailer define the PDF file layout as byte offsets to these indirect PDF objects in the body, as well as identifying the root node of the tree of PDF objects. Indirect PDF objects can be randomly accessed in the file, based on their file offsets. All PDF syntax is case-sensitive and is often referred to as "COS syntax" [9].

PDF defines 9 basic object types: integer, real number, string, Boolean (using keywords '`true`' and '`false`'), names (beginning with SOLIDUS (2Fh) '`/`'), the special null object, arrays (enclosed in '`[`' and '`]`'), dictionaries (enclosed in '`<<`' and '`>>`'), and streams. PDF dictionaries represent hash maps and are defined by key/value pairs, where each key is a case-sensitive PDF name object and key values may be any other type of PDF object. PDF objects may be expressed directly, meaning they written are inline, or indirectly referenced via their object and generation numbers and the single letter keyword '`R`'. PDF streams extend dictionaries to additionally store a sequence of bytes (such as image data) between the '`stream`' and '`endstream`' keywords, Streams must always be referenced indirectly.

```
110 0 obj
<< /BitsPerComponent 8 /ColorSpace /DeviceRGB
/Filter [ /DCTDecode ] /Height 1351 / Interpolate
false /Subtype /Image /Type /XObject /Width 1590
/Length 112 0 R >>
stream
… JPEG compressed binary data not shown …
endstream
endobj
112 0 obj
456783
endobj
```

This short example fragment shows two indirect PDF objects that would be listed in the cross-reference table with their file offsets to the start of the "`110 0 obj`" and "`112 0 obj`" lines respectively. Additional but unnecessary whitespace is shown here solely for clarity. Object 110 is a stream object (as indicated by the '`stream`' and '`endstream`' keywords) with the stream dictionary between the standard dictionary start and end tokens ('`<<`' and '`>>`'). Keys within PDF dictionaries may occur in any order and

[8] https://www.amazon.com/PDF%E3%83%AA%E3%83%95%E3%82%A1%E3%83%AC%E3%83%B3%E3%82%B9%E7%AC%AC2%E7%89%88_Adobe-Portable-Document-Format-Version/dp/4894713381

[9] COS stands for "Carousel Object Syntax" with "Carousel" the original Adobe codename for what later became Acrobat.

the example above shows direct Boolean, integer, and a single element array object as key values. Object 112 is a simple integer object that is indirectly referenced via the `/Length` key in object 110.

The PDF specification defines all valid PDF objects, the permitted types for every dictionary key and array element, data integrity and relationship rules, and a description of what every PDF object is used for. The Arlington PDF Model encodes all DOM information, excepting the usage descriptions, into an easy-to-process machine-readable text-based format.

III. Related Work

The idea of a machine-readable definition of PDF has been discussed for many years. At PDF Association technical events during 2012-2013 various speakers espoused their ideas [9][10][11], as well as at conferences focusing on more specific applications of PDF [12]. Within the ISO working group responsible for PDF (ISO TC 171 SC 2 WG 8), ad-hoc groups were established in early 2012 to evaluate several options proposed by various working group experts, however no progress was made.

Several machine-readable definitions of PDF already exist:

- Adobe DVA (Dictionary Validation Agent) is the underlying proprietary definition used by the "syntax check" preflight feature shipped with Adobe Acrobat, which covers both the PDF object model and content streams, and is itself expressed in PDF COS syntax using custom PDF extensions ([9], Figure 2). Although DVA implies it is a complete and accurate model of PDF 1.7 (as defined by ISO 32000-1:2008), the technical heritage of DVA's development is unknown.

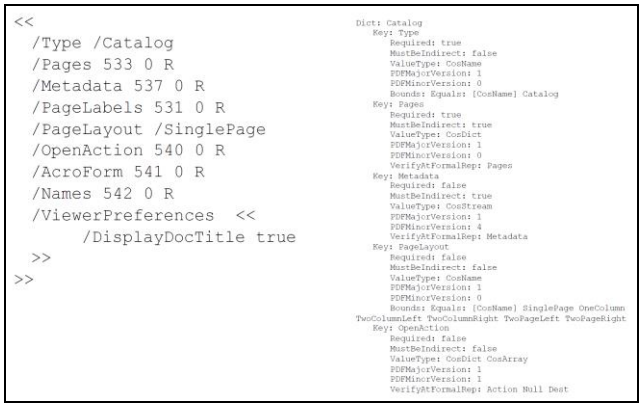

Figure 2: PDF Document Catalog dictionary expressed in Adobe DVA [9]

As the DVA definition is proprietary, additional tooling is required to extract the data into a processable form, so converting and validating DVA was considered at least as much effort as defining a new and open model directly from the PDF specification. As described later (see section V *Model Validation*), a comparison of the Arlington PDF Model to Adobe DVA highlighted a significant number of differences and validated our decision to rigorously establish the new Arlington PDF Model directly from the PDF 2.0 ISO specification source document.

- The veraPDF model ([13]) and model DSL syntax ([14]) were developed as part of the Java-based open-source PDF/A validator. This XText-based DSL ([21]) focuses on the needs for formal ISO 19005 PDF/A file validation and is centered on supporting targeted Java code generation rather than accurately expressing the entirety of the PDF specification. Many details from the core PDF specification are not captured in the veraPDF model as they are not relevant to the PDF/A subset used for archiving and long-term preservation.
- The Levigo "pdf-formal-representation" is another XText-based DSL for describing PDF objects in the PDF specification in a similar manner to the Arlington PDF Model (Figure 3). This framework has not been updated in 7 years and the DSL definition for PDF is not available. However, the design concepts of Levigo's DSL provided confirmation for aspects of the Arlington PDF Model design decisions, such as the choice of some data fields. The Levigo DSL also does not attempt to define any data integrity or complex relationship rules.

```
array-mapped FileIdentifier (pdf-1.1) {
    0 -> PermanentIdentifier required byte-string;
    1 -> UpdatedIdentifier required byte-string;
}

// PDF32000, 7.11.3, table 44
dict-obj FileSpecification ( pdf-1.1 ) {
    /Type name must-be /FileSpec ;
    /FS as FileSystem name ;
    /F text-string ( pdf-1.7 ) ; // FIXME is text-string appropriate for string
    /UF text-string ( pdf-1.7 ) ;
    /DOS byte-string ;
    /Mac byte-string ;
    /Unix byte-string ;
    /ID as FileIdentifier array-mapped FileIdentifier ;
    /V as Volatile bool default false ( pdf-1.2 ) ;
    /EF as EmbeddedFileStream mapping-dict < dict-obj EmbeddedFileStream > ( pdf-1.7 ) ;
    /RF as RelatedFile mapping-dict < array < object > > ( pdf-1.3 ) ;
    /Desc as DescriptiveText text-string ( pdf-1.6 ) ;
    /CI as CollectionItem dict-obj CollectionItem ( pdf-1.7 ) ;
}

// PDF32000, 7.11.4.1, table 45
stream-obj EmbeddedFileStream extends common.StreamDictionary ( pdf-1.3 ) {
    /Type name must-be /EmbeddedFile ;
    /Subtype name ;
    /Params dict-obj EmbeddedFileParameters ;
}

// PDF32000, 7.11.4.1, table 46
dict-obj EmbeddedFileParameters ( pdf-1.3 ) {
    /Size int ;
    /CreationDate date ;
    /ModDate date ;
    /Mac dict-obj MacOSFileInformation ;
    /CheckSum text-string ;
}

// PDF32000, 7.11.4.1, table 47
dict-obj MacOSFileInformation ( pdf-1.3 ) {
    /Subtype int ;
    /Creator int ;
    /ResFork as ResourceFork dict-obj common.StreamDictionary ;
}
```

Figure 3: Levigo's DSL describing PDF objects [10]

Each of these alternate machine-readable definitions has been developed for implementation-centric use-cases. We also considered XText DSL tooling as too heavy and Java-centric, and not conducive to supporting agile research and the broadest possible set of use-cases. As a result, establishing an entirely new model directly from the official master PDF specification would be the same (or less) effort than attempting to confirm and then extend these models. In

addition, a data format could be chosen to maximize accessibility of the model.

A review of existing schema frameworks for other file formats including various XML schemas (e.g., DFD, DTD, XSD, Relax NG), KSY ([18]), DFDL and Apache Daffodil ([19], [20]) provided further some insight into what a machine-readable definition of a complex file format like PDF would need to support and shortcomings to avoid. None of these alternative frameworks can directly support PDF so the creation of a new specialized grammar was required, with strong typing, syntactic and semantic rules, and arbitrarily complex data integrity rules in a concise and easily digestible format.

As a result, the Arlington PDF Model was developed directly from the PDF 2.0 ISO 32000-2:2020 specification master document using text-based structured data and a small custom grammar of predicates.

IV. Methodology

A. Goals of the Arlington PDF Model

The motivation for Arlington PDF Model began with a few explicit design goals:

- To encode the obvious data fields identifiable in each Table in ISO 32000, included key name/array index, the permitted type(s), the version of PDF when a key/array element was introduced, the version of PDF when a key/array element was deprecated or removed, whether a key was required, had to be an indirect reference or a direct object, the optional default value, the set of possible values and the set of pointers (links) to other referenced PDF objects. This was also supported by the previous Levigo DSL work (see section **Error! Reference source not found.** ***Error! Reference source not found.*** **Error! Reference source not found.**). Beyond this we recognized the need for flexibility to encode data integrity and relationship rules.
- The data format chosen needed to be very light-weight and independent of platform and programming languages. Ideally something that could be directly easily processed and queried with Linux commands or simple scripts. A text format was strongly preferred over binary (such as an SQL database or binary serialization). In addition, each PDF object definition should be holistic with all keys and associated data immediately identifiable.
- A simple means for data entry and correction, without the need to develop GUIs or generate reports.
- The data structure should very closely represent the core concepts expressed in the tables in ISO 32000 and without any additional interpretation added. If ISO 32000 was ambiguous then this needed to be directly reflected in a "specification-derived" grammar!
- Support for PDF dictionary and PDF array objects should be seamless, including support for map-style objects with arbitrary key names or an open-ended number of array elements.
- In addition to the basic object types formally defined in ISO 32000, several other types occur regularly (e.g., bitmask, name-tree, number-tree, matrix, rectangle). By implicitly incorporating these additional specialized types into the Arlington PDF Model, the encoding of various data integrity rules could be greatly simplified. For example, the general requirement that a rectangle be 4 floating point numbers representing a pair of (x_1, y_1) (x_2, y_2) coordinates of diagonally opposite corners should not need to be encoded for every occurrence of a rectangle in the PDF DOM, but individual requirements whether a specific rectangle can be degenerate (i.e., of zero height and/or width) or within the bounds of some other rectangle may need to be captured.

B. Answering research questions

Not only is a specification-derived, machine-readable specification of PDF useful for software developers in industry, but it is also highly applicable to research. By having a machine-readable structured data description of "ground truth" PDF that is easy to query, various research questions can be answered that are otherwise almost impossible to reliably extract directly from the PDF specification document.

Within the SafeDocs research program, researchers were able to access the author as a PDF expert and asked various research-oriented questions. With a machine-readable definition of the PDF specification, many such questions could be answered:

- List all valid combinations of PDF object types that the specification defines. This further leads to the identification of all possible valid combinations of adjacent token pairings enabling an optimized full-coverage lexical test that ensures token delimiter processing is fully supported only to the degree required by PDF and is not more permissive. Currently the "Compacted Syntax PDF" ([16]) test developed under SafeDocs covers all possible combinations of token and character pairings;
- Capture of subtle wording that is easily overlooked. For example, a recent Blackhat 2020 presentation ([17]) made incorrect statements about the PDF file format that the Arlington PDF Model could have identified and prevented. This includes that object streams may self-reference, when all PDF specifications have explicitly stated "… *consists of a set of streams whose Extends links form a directed acyclic graph*". The Arlington PDF Model captures and encodes the subtle but explicit acyclic requirement that was missed in Table 16 of ISO 32000.
- List occurrence counts for key names across all dictionaries to identify duplicate or heavily overloaded (and potentially ambiguous) key names. A simple example is identifying all occurrences of a `/N` key. It is very time-consuming to search through all occurrences

of the uppercase letter “N” in the PDF specification (noting that the specification only uses the leading SOLIDUS (/) in example PDF fragments but not in prose). However, with the Arlington PDF Model, a simple Linux `grep` regex instantly identifies 19 occurrences in specific objects as well as their types, when they were introduced, etc. (Figure 4):

```
$ grep -P "^N\t" *.tsv
3DNode.tsv:N                        string       1.7                  TF
ActionNamed.tsv:N                   name         1.2                  TF
Appearance.tsv:N                    dictionary;stream         1.2
AppearancePrinterMark.tsv:N         dictionary;stream         1.4
AppearanceTrapNet.tsv:N             dictionary;stream         1.3      2
Bead.tsv:N                          dictionary   1.1                  TF
BeadFirst.tsv:N                     dictionary   1.1                  TF
CollectionField.tsv:N               string-text  1.7                  TF
FunctionType2.tsv:N                 number       1.3                  TF
ICCProfileStream.tsv:N              integer      1.3                  TF
LinearizationParameterDict.tsv:N    integer      1.1                  TF
MediaClipData.tsv:N                 string       1.5                  FA
MediaClipSection.tsv:N              string       1.5                  FA
ObjectStream.tsv:N                  integer      1.5                  TF
Projection.tsv:N                    number       1.6                  fr
RenditionMedia.tsv:N                string-text  1.5                  FA
RenditionSelector.tsv:N             string-text  1.5                  FA
Target.tsv:N                        string-byte  1.6                  fr
UserProperty.tsv:N                  string-text  1.6                  TF
$
```

Figure 4: PDF objects with /N keys

- Identify PDF objects that do not require a `/Type` or `/Subtype` key and are thus candidates for “schizophrenic objects”, where a single object in a PDF file might contain two or more distinct logical objects in the PDF DOM using merged dictionaries. PDF permits this for a specific situation (see clause 12.5.6.19, [5]) however unexpected occurrences with other PDF objects might cause memory management issues (e.g., use after free) for PDF software;
- Assist in validating results from PDF tooling, before incorrect conclusions are made. For example, the “`pdffonts`” command line utility provided by both Poppler ([35]) and Xpdf ([36]) distributions can report different results for a given PDF file. By using the Arlington PDF Model, all possible paths to font dictionaries can easily be identified and it can then also be determined that both tools will not identify fonts that are themselves resources of Type 3 `CharProc` glyph description content streams. Such detailed knowledge may have altered the conclusions in prior work such as [37].
- Identify dictionary candidates based on a set of ambiguous key names for an isolated snippet of PDF (such as a single dictionary). For example, what kinds of dictionary object might the following of just objects 52 and 53 represent?

```
52 0 obj
<</R[146 522 153 542]/N 53 0 R/P 51 0 R/T 570
0 R/V 53 0 R>>
endobj
53 0 obj
<</R[352 570 359 590]/N 52 0 R/P 51 0 R/T 570
0 R/V 52 0 R>>
endobj
```

Because all the key names in both objects 52 and 53 are just single letters, manually searching the PDF specification document is impractical. The answer in this case is a Beads dictionary, but this can otherwise only be practically determined if object 51 is also available and is recognizable as a Page node with a /B key array elements referencing objects 52 and 53.

- Identification of all PDF name objects explicitly specified in the specification. Such a list can then be compared against source code or a corpus to determine which features are implemented, or partially implemented, in a code base or present in a corpus;
- Identify all valid references to or from a specific PDF object. This defines all incoming edges to a specific vertex (object) in the directed graph that is the PDF DOM. Interactive 3D and virtual reality visualizations of an early iteration of the Arlington PDF Model for each PDF version have been created at https://safedocs.pdfa.org/, however the sheer number of edges and vertices makes gaining insights directly from the raw data difficult (Figure 5):

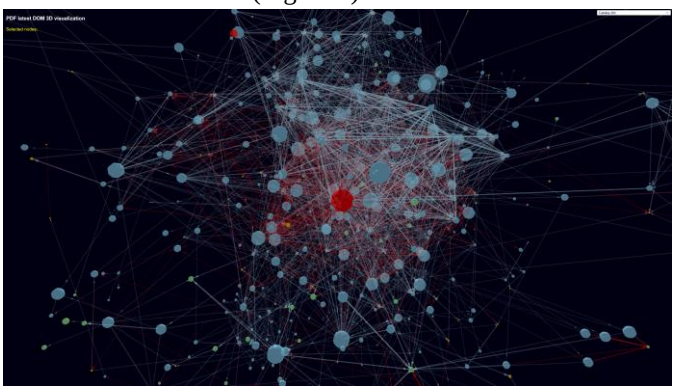

Figure 5: Interactive 3D visualization of the PDF 2.0 object model

Application of a specification-derived machine-readable definition of PDF enables new capabilities across a broad spectrum of PDF-based research.

C. Initial creation

The public ISO specification of PDF 2.0 is a 1,000-page PDF document created from a master Microsoft Word™ document that utilizes a stylesheet for consistent formatting. The definition of PDF objects (specifically, dictionary objects and all their keys) are primarily in over 400 3-column tables that use consistent styling and structure, as shown in Figure 6 below.

Dictionary key names are described in the first column but without the leading SOLIDUS (2Fh) ‘/’. The “Type” column lists one or more permitted PDF object types. The “Value” column starts with bracketed italic statement regarding Required vs Optional and any version-specific information. Within prose, key names are formatted as bold and key values are italic to assist understanding. The “Value” column may also contain a “Default value” statement as well as other freeform prose. Many tables are also lengthy and span across page breaks.

Key	Type	Value
V	integer	(*Required*) A code specifying the algorithm to be used in encrypting and decrypting the document: *0* An algorithm that is undocumented. This value shall not be used. *1* (*Deprecated in PDF 2.0*) Indicates the use of 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) with a file encryption key length of 40 bits; see below. *2* (*PDF 1.4; deprecated in PDF 2.0*) Indicates the use of 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) but permitting file encryption key lengths greater than 40 bits. *3* (*PDF 1.4; deprecated in PDF 2.0*) An unpublished algorithm that permits file encryption key lengths ranging from 40 to 128 bits. This value shall not appear in a conforming PDF file. *4* (*PDF 1.5; deprecated in PDF 2.0*) The security handler defines the use of encryption and decryption in the document, using the rules specified by the **CF**, **StmF**, and **StrF** entries using 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) with a file encryption key length of 128 bits. *5* (*PDF 2.0*) The security handler defines the use of encryption and decryption in the document, using the rules specified by the **CF**, **StmF**, **StrF** and **EFF** entries using 7.6.3.3, "Algorithm 1.A: Encryption of data using the AES algorithms" with a file encryption key length of 256 bits.
Length	integer	(*Optional; PDF 1.4; only if V is 2 or 3; deprecated in PDF 2.0*) The length of the file encryption key, in bits. The value shall be a multiple of 8, in the range 40 to 128. Default value: *40*.
CF	dictionary	(*Optional; meaningful only when the value of* ***V*** *is 4 (PDF 1.5) or 5 (PDF 2.0)*) A dictionary whose keys shall be crypt filter names and whose values shall be the

Figure 6: Example extract from ISO 32000-2

Pagination artifacts and the ISO publication process make extracting structured data for all PDF object definitions from the official ISO PDF publication extremely difficult. However, by using the master Microsoft Word™ document and knowledge of the stylesheet usage, macros and advanced searching could be used to extract information efficiently and reliably from the PDF specification master document.

Through a semi-automated process combined with significant manual checks and edits, as well as correcting several editorial formatting and style fixups to the ISO 32000-2 master document, the core data on all PDF objects was transformed into a structured LibreOffice Calc spreadsheet.

The structured spreadsheet definition allowed easy bulk data entry and navigation, including simple automation, data validation and bulk corrections using spreadsheet scripting. With one worksheet for each PDF object (i.e., one worksheet for each dictionary, array, or stream object), the spreadsheet soon exceeded the capabilities of Microsoft Excel™ and LibreOffice Calc was required to be used.

Note that not all 512 objects in the current Arlington PDF Model correlate one-to-one to a Table in the ISO PDF standard. A significant number of PDF objects were found to be defined inline in prose-based descriptions (e.g., Table 100, ISO 32000-2:2020, [5]) or through simple statements (e.g., "an array of RGB values"). The number of such inline object descriptions necessary to define the entire PDF DOM was unexpected.

A script was developed to export each worksheet to a single Tab-Separated Value (TSV) file[10], creating a filesystem-based database of all PDF objects. Further processing, validation and proof-of-concept applications could then be developed by processing the textual TSV data file set.

TSV files are platform independent, text-based, directly supported by Python and Perl, easily supported in other programming languages, are directly visualized in GitHub, and can be easily processed by Linux CLI such as `cut`, `grep`, `sed`, etc. TSV data manipulation is also directly supported by existing 'big data' tools such as the eBay `tsv-utils` ([33]) and GNU `datamash` ([34]).

The final Arlington PDF Model data structure is shown in Table 2 and comprises 12 fields (columns) with each row in a TSV file representing a single key in a dictionary or stream. or an array element. Each TSV file is the entire standalone definition for a single PDF object (i.e., dictionary, array, or stream). Many fields in the TSV data (Table 2 below) closely match the information in the Tables in the PDF specification as shown in Figure 6:

#	Column Name	Description
1	`Key`	key in dictionary, or integer index as array index. ASTERISK (*) represents a wildcard and means "any key/index".
2	`Type`	one or more Arlington types separated by SEMI-COLON.
3	`SinceVersion`	version of PDF this key was introduced
4	`DeprecatedIn`	version of PDF this key was deprecated in. Empty if not deprecated.
5	`Required`	whether key or array element is required.
6	`IndirectReference`	whether the key is required to be an indirect reference (TRUE/FALSE) or if it must be a direct object (`fn:MustBeDirect`)
7	`Inheritable`	whether the key is inheritable through the page tree (TRUE/FALSE).
8	`DefaultValue`	optional default value of key.
9	`PossibleValues`	list of possible values. For dictionary keys that must have a specific value, this will be a choice of just 1.
10	`SpecialCase`	predicates defining additional relationships
11	`Link`	name(s) of other TSV files for validating the values of this key for dictionaries, arrays, and streams.
12	`Notes`	free text for arbitrary notes.

Table 2: Summarized Arlington TSV field definitions

As the richness of the data being captured increased (i.e., individual cells in the spreadsheet became lengthy), and repetitive edits to multiple worksheets were required, and the level of TSV data validation increased, it was soon easier and more efficient for data updates to be done directly into TSV files using text and TSV-aware editors.

This is the specification-derived, machine-readable TSV-based data file definition of PDF 2.0 that is now known as the Arlington PDF Model[11].

[10] TSV was initially selected over CSV as GitHub displays TSV natively in tabular form.

[11] This name is in keeping with other PDF Association assets which use place names, such as the Isartor Test Suite and Matterhorn Protocol. Arlington is the location of DARPA in the USA.

D. *Expressing object and data integrity relationships*

Within the SafeDocs research community, discussions were held to decide on possible methods to express more complex object and data integrity relationships that did not directly map to the columnar TSV fields. SafeDocs researchers intend to transcode all relationships defined in the Arlington PDF Model (i.e., all relationships documented in the PDF 2.0 specification) to their DDLs with a goal of formally identifying new or unstated requirements, ambiguities in PDF, etc. via their formal methods. Options considered included incorporating fragments of Python, JavaScript, or Lua, as well as migrating to a full DSL system (see section IV.E *Alternate DSL representation* below).

Desired attributes for expressing PDF object and data integrity relationships included:

- Equal support for PDF readers and PDF writers;
- Clarity and readability, especially when side-by-side with the ISO 32000 standard or when read aloud;
- Maintainability – as the process is iterative, the full set of predicates is not known a-priori and is subject to change;
- A single way to express each kind of unique relationship;
- Direct representation of wording from the PDF specification, irrespective of how poor the choice of those words might be. There was to be no reinterpretation of the specification wording necessary to codify relationships!
- Support for future extensions to the Arlington grammar, such as well-known (acceptable) malformations, proprietary extensions, new SafeDocs-inspired annotations, etc.
- Concise and simple to parse.

A decision to use a simple purely declarative style custom grammar of predicate statements was made as other options implied either workflows that did not suit all SafeDocs approaches, or heavy implementation and usage burdens.

Early versions of the declarative function grammar attempted to use special prefix symbols (e.g., '`#`', '`$`'), however this approach conflicted with various capabilities across the spreadsheet applications. As a result, all custom declarative functions in the Arlington PDF Model now begin with the fixed text string "`fn:`" (which does not exist in any PDF specification) and are factual statements (predicates) that directly mirror phrasing from the PDF specification.

Certain predicates can also take parameters such as a condition (e.g., presence or absence of another key; a specific value of a key (represented by a key name prefixed with '`@`')), a PDF version, bit positions, etc. Predicates may also be nested, combined in simple logic statements (just OR (`` `||` ``) and AND (`` `&&` ``)), or express statements that need to be evaluated (using the generic `fn:Eval` predicate). Full details of the Arlington PDF Model grammar and the custom declarative function predicates are in GitHub [1].

Arlington PDF Model predicate examples
`fn:IsRequired(fn:SinceVersion(2.0) || fn:IsPresent(Encrypt))`
`fn:MustBeDirect(fn:IsPresent(Encrypt))`
`fn:Ignore(@SubFilter==ETSI.RFC3161)`
`fn:IsRequired(fn:NotPresent(DParts))`
`fn:Eval(fn:ArrayLength(Widths)==(@LastChar-@FirstChar+1))`

Table 3: Sample Arlington PDF Model predicate statements

Currently there are 40 predicates that are easily identifiable and matched to the equivalent wording in the PDF specification. The total number of unique predicates required to encode all object and data integrity relationships in ISO 32000 is not known.

E. *Alternate DSL representation*

After the initial data creation but before iterating to comprehensively capture complex relationships, a review and prototyping exercise was conducted on switching from the columnar TSV structured data format to a more freeform and expressive DSL such as those supported by XText (the Levigo DSL [15] was considered as easy to read and comprehend).

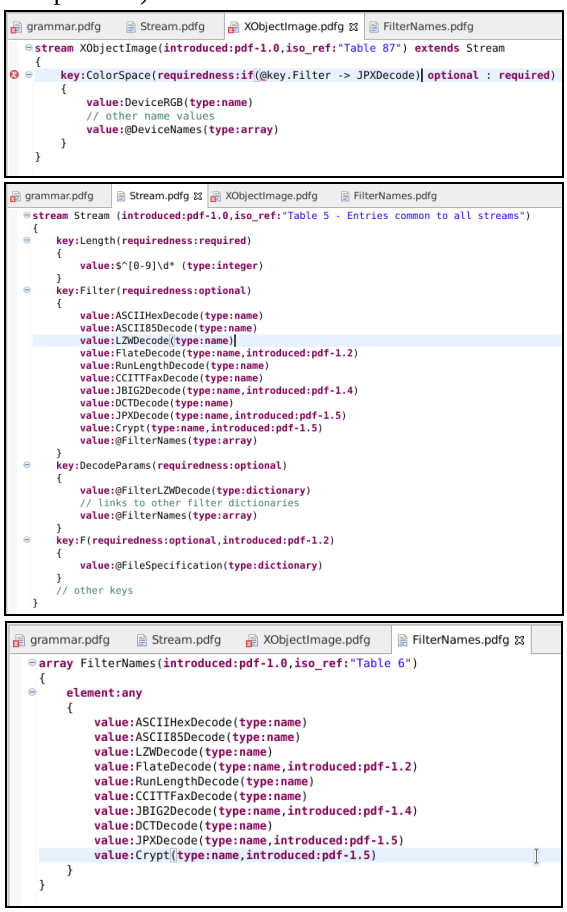

Figure 7: Prototype Arlington using XText DSL

A custom DSL and associated tooling were thought to provide benefits such as a conciseness, syntax highlighting, auto-completion, a richer grammar, and an ability to capture "object hierarchies". Figure 7 shows how a 'base class' stream object might be expressed in an Arlington PDF DSL and how other specialized stream objects (e.g., Image XObject) could extend the definition, rather than needing to repeat data across multiple TSV object definitions as is currently required.

However, the disadvantages were far more significant: the grammar was still rapidly evolving so constant maintenance of both a DSL and supporting toolchain would be a significant overhead; the heavier tooling would place a greater burden on current research use within SafeDocs (no team was using Java); and the ability to quickly conduct investigations using Linux CLI and the 'big data' utilities would no longer be possible.

As a result, the Arlington PDF Model remains in a structured TSV text format, however the author is still open to considering a DSL in the future.

V. Model Validation

A. Benchmarking to Adobe DVA

A comparison of the Arlington PDF Model (based on ISO 32000-2:2020, [5]) against the Adobe DVA definition (based on ISO 32000-1:2008, [3]) was conducted to validate the Arlington PDF Model data. Errors were identified across both models but after manually correcting errors in Arlington, 118 objects (from a total of 504 Arlington objects at the time of comparison) with 623 technical differences with Adobe DVA remained. Changes since PDF 1.7 and related to PDF 2.0 were fully expected (e.g., new keys, deprecated keys, previously optional keys that are now required in PDF 2.0, etc.) but an unexpectedly large number of differences in DVA were still identified.

Differences included missing keys, the presence of undocumented keys, differences in required/optional data, differences in data types, and differences in versioning:

- Spelling mistakes in key names (e.g. "`/GotToRemote`" instead of "`/GoToRemote`");
- Undocumented additional types for some keys, such as DVA permitting both a dictionary and an array of dictionaries where ISO 32000-1:2008 (PDF 1.7) permits only arrays. This might be assumed as a permissive degenerate condition for situations when the dictionary would be a single element array. It is unknown whether such issues were triggered by extant data or previously incorrect implementations;
- Several common and well-understood PDF keys are specificied to exist across multiple different objects. However DVA defined that some such keys were further permitted in additional objects in the PDF DOM that do not match any legacy or current PDF specification;
- Differences between required and optional keys;
- Previously unknown and undocumented keys;
- Differences in version information.

A report listing all differences between the Arlington PDF Model and DVA has been shared with Adobe. Note that as DVA is only used for Adobe's "syntax check" preflight in Acrobat, this does not necessarily indicate that Adobe's PDF parser permits such errors.

B. Validating with extant data

Another means of validating the Arlington PDF Model is to compare to the de-facto specification defined by extant data. A C++ proof-of-concept (PoC) application was developed to perform a basic comparison of a PDF files' objects against the Arlington PDF DOM. The C++ PoC can report various errors including:

- When the type of a key or array element is incorrect;
- When the value of a required key or array element is incorrect. This error also discovered a several extant malformations, obviously caused by careless typos in PDF writers, such as:

```
Error: wrong value: Type
(XObjectImageSoftMask.tsv) should be: NAME
[XObject] and is name (XObjcect)
```

- When required keys or array elements are not present. As PDF 2.0 changed some previously optional font-related keys to required, legacy extant data may get falsely flagged until the PoC application is upgraded to perform detailed PDF version checks;
- When an object that is required to be indirect object is a direct object or vice-versa;

As a direct result of analyzing these reported errors across a corpus of over 200,000 PDFs, it was noted that a key required by all PDF specifications was almost never present. This was raised to ISO TC 171 SC 2 WG 8 (the working group responsible for ISO 32000) and, as a result, the latest ISO 32000-2:2020 PDF 2.0 document has been changed to reflect this de-facto PDF specification[12].

C. Self-consistency

In addition to benchmarking against extant data, aspects of the Arlington PDF Model can also be internally validated.

Every PDF DOM object defined in an Arlington file set must be logically reachable from the PDF trailer by following the "Link" fields. This applies to the file set for each PDF version that can be derived from the main PDF 2.0 definitions by excluding PDF objects introduced by later PDF versions, thus validating the PDF version information within the PDF specification. This is also visible in the 3D/VR visualizations described in section IV.B *Answering research* above, as isolated vertices (PDF objects) without any edges (links).

Other self-consistency model checks include requiring that all streams are indirect-references; that all dictionary,

[12] The `/Subtype` key for thumbnail images is no optional in PDF 2.0.

array and stream objects have a link to another Arlington object definition; that "DefaultValue" and "PossibleValue" values match the defined type(s); that the "SinceVersion" PDF version is the same or later than any linked object definition; and that key names and array indices are unique.

This process identified both data entry errors in the Arlington PDF Model and inconsistencies in the PDF 2.0 ISO specification that have since been corrected. See IX *Appendix* below for more details.

VI. Additional Benefits

A. Reflections

Creating the Arlington machine-readable definition of PDF directly from the ISO specification document was a methodical iterative process. Initially there were some minor design choices on how to efficiently and effectively encoding the 'static' data represented by the "PossibleValues" and "Link" fields, but once about half of the PDF objects had been entered this churn had resolved and the field design has remained stable.

Once the model achieved a certain size and complexity, additional formality was needed to define exactly what each column (field) could contain and precisely how that was unambiguously expressed in the Arlington model grammar. This included recognizing a few unique situations in the PDF specification:

- We neglected to notice early during creation that there is a requirement that some keys must always be direct objects. This was resolved by creating the `fn:MustBeDirect` predicate in the "IndirectReference" field.
- Deprecation not only occurs for key values, but can also occur for individual types for multi-typed keys (such as Graphic State Parameter /BM key where the array form was deprecated in PDF 2.0);
- All streams must be indirect objects (by definition) so the "IndirectReference" field is thus required to be multi-valued to support keys/array elements that allow stream objects as well as other object types which might be direct objects;

Identifying and resolving these situations was relatively simple once we had migrated to TSV data. Simple Python scripts and editor macros could perform bulk data changes which could then be confirmed by self-consistency validation.

We also identified ambiguous terminology that is impossible to codify consistently across different implementations when using traditional programming languages. Use of the terms "*meaningless*" and "*meaningful*"[13] are not adequately defined with examples including "*Optional for images that use the JPXDecode filter, meaningless otherwise*" and "*… meaningful only when the value of **V** is 4*". Other terminology choices, such as "*shall*", "*shall not*" or "*ignore*", are far more explicit and thus easier to codify repeatedly. Questions arise for parsing about whether a "*meaningful*" key with an incorrect type, or a known type but unknown value should generate validation syntax warnings or be rejected as a runtime error. The Arlington PDF Model captures the exact phrasing used in the specification without re-interpretation by defining custom predicates `fn:IsMeaningful` and `fn:Ignore`.

We continue to iterate through the PDF specification document using a methodical approach. For example:

- searching for every occurrence of "`integer`"[14] and "`number`" to identify explicit data range constraints;
- searching for every occurrence of "`array`" to identify explicit array size constraints (encoded using the `fn:ArrayLength` predicate);
- searching for every occurrence of "`string`" to identify the correct PDF string sub-type (`string`, `string-ascii`, `string-byte`, or `string-text`) and any explicit length constraints (encoded using the `fn:StringLength` predicate);

Every iteration identifies numerous improvements to the PDF ISO specification which, if each were considered in isolation, may seem minor but in totality have significantly contributed to a far more precise and rigorous definition for valid PDF file syntax. See section IX Appendix below for a list of all changes.

B. PDF Observatory

Misspelt PDF keys have been reported for many years ([23]). Elasticsearch's Term Suggester also calculates the Damerau-Levenshtein distance between two words[15]. When combined with the Arlington PDF Model, the SafeDocs "PDF Observatory" system ([32]) now has the unique capability to automatically identify malformations in any corpus. Both the occurrence count and creator/producer metadata for each malform is also returned and can be visualized, allowing impact and attribution to be quickly assessed. This rapidly highlighted several real-world PDF malformations such as misspelt or incorrectly capitalized PDF key names including `/Type` vs. `/type`, `/Subtype` vs. `/SubType`, `/XObject` vs. `/XObjcect`, and `/BlackIs1` vs. `/Blackls1` (uppercase i vs. lowercase L).

C. PDF version complexity

While creating the 3D/VR visualizations for each PDF version, it was realized that the Arlington PDF Model can also be used as a basic metric of PDF format complexity. By tabulating all keys and objects introduced and deprecated in each version of PDF, a histogram visualization creates a simple "net specification growth" metric for each version of PDF over time (Figure 8):

[13] See https://github.com/pdf-association/pdf-issues/issues/6

[14] See https://github.com/pdf-association/pdf-issues/issues/15

[15] This distance represents the minimum number of single-character edits (insertions, deletions or substitutions, including change of case) or transposition of characters required to change one word into another.

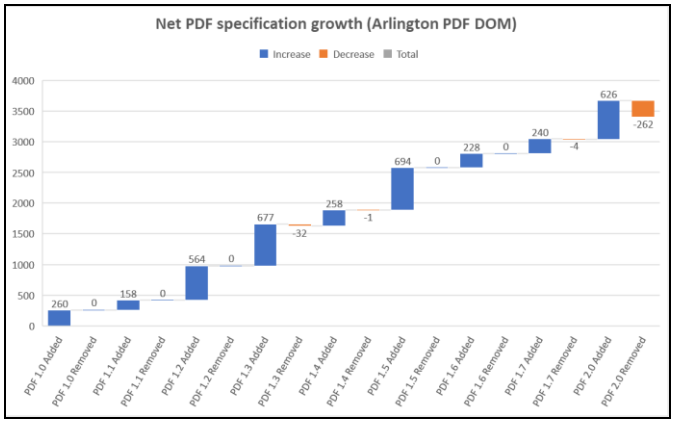

Figure 8: Net PDF specification growth

Note that this metric only reflects distinct parser point changes (i.e., introduction or removal of a dictionary key or array element and associated objects) - it does not account for the "degree of difficulty" required to fully implement the associated business rules. For example, PDF 1.4 introduced the transparent imaging model which is widely regarded as the single most significant and complex change to PDF, yet this is not reflected in the number of new objects or keys added with PDF 1.4. Possibly as the Arlington PDF Model expands to include richer relationships and data integrity rules (predicates), a form of weighted histogram may more closely match industry expectations.

VII. Future Work

The Arlington PDF Model currently describes the full PDF 2.0 object model and intra- and inter-object relationships. At the time of writing, not every relationship between objects has been captured, which implies the set of declarative functions (predicates) may not be finalized. A manual and methodical review by file format experts continues to review ISO 32000-2:2020 to identify explicit requirements ("shall" and "should" statements) as well as legacy requirements that may not use ISO terminology.

The original motivation for the Arlington PDF Model was to define machine-readable object definitions for all objects that can occur in the PDF DOM, however the model could be extended to include file layout, lexical analysis and tokenizing requirements that are precursor steps to DOM construction for a parser.

Future extensions to the Arlington PDF Model might also include:

- **Legacy PDF features.** Features from the Adobe PDF 1.0 to Adobe PDF 1.7 specifications have not been methodically captured, which means the Arlington PDF Model may not yet fully reflect older legacy extant data;
- **PDF subset restrictions**. Additional restrictions and limitations defined across the various ISO PDF subset standards have not been captured, such as for PDF/X (ISO 15930), PDF/VT (ISO 16612), PDF/UA (ISO 14289), PDF/E (ISO 24517), PDF/R (ISO 23504), PDF/VCR (ISO 16613) and PDF/A (ISO 19005). Such PDF subset standards can only constrain existing requirements in their associated base PDF standard and cannot contradict any base PDF requirements;
- **Commonly accepted extensions (including malformations).** These are widely documented and implemented extensions, such as Appendix H "Compatibility and Implementation Notes" of all legacy Adobe PDF specifications or in the later Adobe extensions to ISO 32000-1:2008 series of specifications. An example is implementation note 7 "*Acrobat viewers accept the name **DP** as an abbreviation for the **DecodeParms** key in any stream dictionary. If both **DP** and **DecodeParms** entries are present, **DecodeParms** takes precedence*.";
- **Nested Formats.** Arlington does not currently support inter-format or nested format rules. Such an extension might capture additional rules to ensure that PDF object ky values describing images are aligned with the associated nested (embedded) image data, such as width, height, bits/channel, and color space. PDF content streams with their graphical operators and operands are also not currently supported, although they are supported by the Adobe DVA proprietary grammar. Arlington also does not define the small subset of PostScript defined for PDF Type 4 PostScript calculator function streams. Such a definition would have helped identify that [17][16] uses invalid Type 4 PostScript, while also assisting implementations reduce their 'attack surface' by replacing unnecessary full PostScript interpreters with optimized minimal implementations;
- **FDF**. Arlington does not currently support Form Data Fields (FDF) files which are defined in ISO 32000, use the same PDF COS syntax, and share many PDF DOM objects. FDF files can be used to save annotations and form data fields separately from a PDF form and are an important part of a functional PDF ecosystem. XFDF (ISO 19444) is a separate ISO standard which defines FDF equivalent data in an XML tagset and thus could use any of the existing XML schema languages for detailed XFDF file format validation;
- **Specification cross-references**. There are currently no cross references back into the PDF specification from the Arlington data, such as a Table or sub-clause number from within the TSV data files. When the TSV filename is insufficient, the current informal procedure requires identifying an unusual key name or value in a TSV file and searching for that within the specification document. As the ISO PDF specification does not use line numbers, anchoring definitions for PDF objects that are not defined by an explicit table is problematic;

[16] https://i.blackhat.com/USA-20/Thursday/us-20-Mueller-Portable-Document-Flaws-101.pdf#page=20

- **Research extensions.** Researchers in other domains may wish to augment the Arlington PDF Model. For example, security researchers such as [28] may wish to annotate PDF objects and fields where personally identifiable information (PII) or metadata might get exfiltrated, or digital preservation researchers may wish to annotate the model with provenance metadata details from the PDF/A ISO 19005 family of standards or certain legacy malformations critical to understand existing file repositories.

This may seem like a long list of extensions, however the Arlington PDF Model already defines the vast bulk of pages in the ISO PDF 2.0 specification which define all PDF objects and the entire PDF DOM. Only relatively few pages in ISO 32000 need to be read before the Arlington PDF Model can be understood.

Extensions to the Arlington PDF Model need to retain the flexibility and ease of processing that is currently available via Linux commands and 'big data' utilities. For example, if key name alternatives (such as misspellings) were added to column 1 ("Key" field) using a new predicate, then simple Linux `grep` and `tsv-utils` commands would become more complex. Alternate design choices include adding an additional column (field) to all TSV files, expanding the existing "SpecialCase" field to use a new predicate, or implementation as a set of diffs, stored externally, which could be applied to an evolving Arlington PDF Model, and thus retain the purity of the specification-derived model.

VIII. Conclusion

This work presents the Arlington PDF Model as the first open access, vendor neutral, comprehensive specification-derived machine-readable definition of all formally defined PDF objects and their intra- and inter-object relationships. This represents the bulk of the latest 1,000-page ISO PDF 2.0 specification and is a definition of the entire PDF DOM. As it is directly derived from the source file of the latest PDF 2.0 specification (ISO 32000-2:2020) and amended by the industry-agreed errata resolutions, this resource establishes a state of the art "ground truth" for future PDF research efforts and implementers.

Conceptually, the structured data model is simple to understand as it closely mimics the ISO PDF specification documents. It does not require deep PDF knowledge, specific domain expertise, or specialist skills in DSLs or DDLs. Expressed as a set of text based TSV files, with 12 data fields and a set of custom declarative functions (predicates), simple Linux CLI commands, pre-packaged 'big data' utilities, and custom scripts can all be used to query the model without a steep learning curve. It is readily accessible to a diverse audience of researchers, implementers, and other interested users.

The Arlington PDF Model has been successfully validated against an alternate proprietary model, as well as a sizeable corpus of extant data files and has been widely shared within the SafeDocs community and the PDF Association's PDF Technical Working Group. It has already highlighted various extant data malformations and triggered multiple changes to the PDF 2.0 specification to reflect the de-facto specification, remove ambiguities, and correct errors. The ongoing refinement of the Arlington PDF Model will undoubtedly continue to identify further issues with ISO 32000-2.

IX. Appendix

List of issues in ISO 32000-2 (PDF 2.0) identified via the Arlington PDF Model. Those issues identified as "post-publication" of ISO 32000-2:2020 are tracked via https://github.com/pdf-association/pdf-issues.

ISO Ref.	Description of Issue	Publication
7.7.2	AA key incorrectly attributed to PDF 1.4, instead of PDF 1.2	ISO 32000-2:2020
12.3.4	Thumbnail Image XObject Subtype key is not used in significant extant data	ISO 32000-2:2020
12.5.6.24	Projection annotation Subtype value never specified	ISO 32000-2:2020
13.6.6	Confusion over ExData key and when it is a required key	ISO 32000-2:2020
12.5.6.2	Which kinds of annotations can have ExData is unclear	ISO 32000-2:2020
12.5.6.6	Table 172 vs 177 RC key have different definitions	ISO 32000-2:2020
12.6.4.18	Table 223 non-standard terminology in Type column for A key	ISO 32000-2:2020
12.5.6.9	Table 181 IC key PDF version confusion	ISO 32000-2:2020
9.6.4	Type3 FontDescriptor - required, conditional required or optional?	ISO 32000-2:2020
7.6	Clause 7.6 and Table 20 "deprecated" wording does not state version	ISO 32000-2:2020
7.6.1	Encryption V in Table 20 says number should be integer	ISO 32000-2:2020
7.6.4.2	Encryption R in Table 21 says number should be integer	ISO 32000-2:2020
7.6.4.2	Table 21 uses "string" but should use "byte string" for many keys	ISO 32000-2:2020
10.6.5.6	Table 132 HalftoneType is number but everywhere else integer	ISO 32000-2:2020
8.11.4.4 Table 100	Optional Content Usage Zoom sub-dict - Min/Max are numbers (unstated)	ISO 32000-2:2020
12.8.2.4 Table 259	Table 259 FieldMDP V key - required statement implies only in PDF 1.5	ISO 32000-2:2020
7.5.5 Table 15	"Required in PDF 2.0" --> "Required by PDF 2.0 and later"	ISO 32000-2:2020
Various	Common use of term "meaningful" is inadequately defined	post-publication
Table 5	Table 5 confusion if Length is optional when F key is present	post-publication
Table 120	Type3 FontDescriptors FontName confusion	post-publication
Table 132	Table 132 Default key description has incorrectly worded requirement	post-publication

ISO Ref.	Description of Issue	Publication
Tables 87 & 143	Soft-mask image dictionary key limitations between Table 143 & Table 87	post-publication
Table 137	Confusion in requirements for CS and C for 3D background	post-publication
Table 47	Collection Subitem: Default is confusing - is it string or nothing?	post-publication
Table 164	Di should be integer, not number	post-publication
13.6.4.3	Does annotation dictionary BM key apply to 3D backgrounds?	post-publication

Table 4: ISO 32000-2 issues identified by Arlington

X. Acknowledgment

The author wishes to acknowledge the support of Duff Johnson, CEO, PDF Association Inc. as well as the support and understanding of Dr. Sergey Bratus in helping us along the SafeDocs journey. Special thanks and appreciation to Roman Toda and Frantisek Forgac of Normex s.r.o. for their perseverance, patience and dedication while developing the rapidly churning early versions of the Arlington PDF Model. In addition, special thanks to Dr. Tim Allison and his team at Jet Propulsion Laboratory, California Institute of Technology for their support and willingness in trying new ideas as part of the SafeDocs "PDF Observatory". Without the many insightful questions from all the SafeDocs researchers, the Arlington PDF Model as well as the many improvements realized in ISO 32000-2:2020 would also not have been possible. Thank you also to Dr. Samuel Weber for shepherding this paper and providing helpful feedback and sage advice on early drafts.

XI. References

[1] Arlington PDF Model, https://github.com/pdf-association/arlington-pdf-model.

[2] John Warnock, "The Camelot Project." Adobe Systems Inc., 1991, [Online]. http://www.eprg.org/G53DOC/pdfs/warnock_camelot.pdf.

[3] *Document management — Portable document format — Part 1: PDF 1.7*, ISO 32000-1:2008.

[4] *Document management — Portable document format — Part 2: PDF 2.0*, ISO 32000-2:2017. Withdrawn. https://www.iso.org/standard/63534.html

[5] *Document management — Portable document format — Part 2: PDF 2.0*, ISO 32000-2:2020. https://www.iso.org/standard/75839.html

[6] Simon Pieters, "Idiosyncrasies of the HTML parser", [Online] https://htmlparser.info/.

[7] RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels", March 1997. [Online] https://tools.ietf.org/html/rfc2119.

[8] "ISO/IEC Directives, Part 2 Principles and rules for the structure and drafting of ISO and IEC documents", Eighth edition, 2018. [Online] https://www.iso.org/sites/directives/current/part2/index.xhtml.

[9] *PDF Validation*, Leonard Rosenthol (Adobe Systems), PDF Technical Conference 2012, Basel, Switzerland. Unpublished.

[10] *PDF Runtime Validation and beyond...*, François Fernandès (levigo solutions gmbh), PDF Technical Conference 2013, Königswinter, Germany. https://youtu.be/-K_yBHw3C0U and https://github.com/levigo/pdf-formal-representation

[11] *PDF Validation*, Leonard Rosenthol (Adobe Systems), PDF Technical Conference 2013, Königswinter, Germany. https://youtu.be/-K_yBHw3C0U

[12] Duff Johnson, "Achieving Canonical PDF Validation" in *Proceedings of the 11th International Conference on Digital Preservation*, Melbourne, Australia, Oct. 2014, p. 5, [Online]. Available: https://www.nla.gov.au/sites/default/files/ipres2014-proceedings-version_1.pdf

[13] VeraPDF Model, https://github.com/veraPDF/veraPDF-model.

[14] VeraPDF Model Syntax, https://github.com/veraPDF/veraPDF-model-syntax.

[15] Levigo PDF formal representation, https://github.com/levigo/pdf-formal-representation.

[16] SafeDocs PDF Compacted Syntax, https://github.com/pdf-association/safedocs/tree/main/CompactedSyntax.

[17] Jens Müller, Dominik Noss, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk, "Portable Document Flaws 101," presented at the BlackHat USA 2020, USA, Aug. 06, 2020, [Online]. Available: https://www.blackhat.com/us-20/briefings/schedule/#portable-document-flaws--20387.

[18] KaiTai Struct. https://kaitai.io/

[19] Data Format Description Language (DFDL), Open Grid Forum. https://www.ogf.org/ogf/doku.php/standards/dfdl/dfdl

[20] Apache Daffodil, https://daffodil.apache.org/

[21] XText, https://www.eclipse.org/Xtext/

[22] Trevor Owens, The Theory and Craft of Digital Preservation., 1st Edition, USA: John's Hopkins University Press, 2018.

[23] G. Endignoux, O. Levillain, and J. Migeon, "Caradoc: A Pragmatic Approach to PDF Parsing and Validation," in *2016 IEEE Security and Privacy Workshops (SPW)*, May 2016, pp. 126–139, https://dx.doi.org/10.1109/SPW.2016.39.

[24] Andreas Bogk and Marco Schöpl, "The Pitfalls of Protocol Design: Attempting to Write a Formally Verified PDF Parser," in *2014 IEEE Security and Privacy Workshops*, San Jose, CA, May 2014, pp. 198–203, https://dx.doi.org/10.1109/SPW.2014.36

[25] J. Müller, D. Noss, C. Mainka, V. Mladenov, and J. Schwenk, "Processing Dangerous Paths," NDSS, p. 16, Feb. 2021, doi: 10.14722/ndss.2021.2310.

[26] C. Carmony, M. Zhang, X. Hu, A. Vasisht Bhaskar, and H. Yin, "Extract Me If You Can: Abusing PDF Parsers in Malware Detectors," presented at the *Network and Distributed System Security Symposium*, San Diego, CA, 2016, https://dx.doi.org/10.14722/ndss.2016.23483.

[27] T. Kuchta, T. Lutellier, E. Wong, L. Tan, and C. Cadar, "On the correctness of electronic documents: studying, finding, and localizing inconsistency bugs in PDF readers and files," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3187–3220, Dec. 2018, doi: 10.1007/s10664-018-9600-2.

[28] S. Adhatarao and C. Lauradoux, "Exploitation and Sanitization of Hidden Data in PDF Files: Do Security Agencies Sanitize Their PDF files?," p. 11, Mar. 2021, [Online]. Available: https://arxiv.org/pdf/2103.02707.pdf.

[29] Ron Brandis and Luke Steller, "Threat Modelling Adobe PDF," Australian Government Department of Defence, Defence Science and Technology Organisation, DSTO-TR-2730, Aug. 2012.

[30] Ian Markwood, Dakun Shen, Yao Liu, and Zhuo Lu, "PDF Mirage: Content Masking Attack Against Information-Based Online Services," Jun. 2017, [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/markwood.

[31] D. Tran and C. Jaiswal, "PDFPhantom: Exploiting PDF Attacks Against Academic Conferences' Paper Submission Process with Counterattack," in 2019 *IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference* (UEMCON), Oct. 2019, pp. 0736–0743, doi: 10.1109/UEMCON47517.2019.8992996.

[32] T. Allison et al., “Building a Wide Reach Corpus,” in LangSec 2020, May 2020, p. 9, [Online]. Available: http://spw20.langsec.org/papers/corpus_LangSec2020.pdf.

[33] eBay tsv-utilities, https://github.com/eBay/tsv-utils.

[34] GNU Datamash, https://www.gnu.org/software/datamash/.

[35] Poppler PDF library, https://poppler.freedesktop.org/.

[36] Xpdf PDF library, https://www.xpdfreader.com/.

[37] A. N. Jackson, “Using Automated Dependency Analysis To Generate Representation Information,” in arXiv:1111.0735 [cs], Nov. 2011, p. 4, [Online]. Available: http://arxiv.org/abs/1111.0735.