LaTeX Tagged PDF Feasibility Evaluation

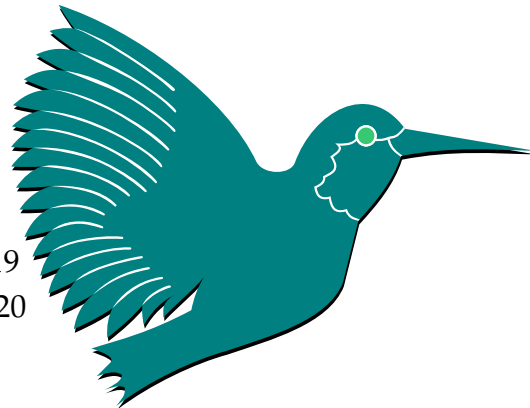LaTeX Project

Frank Mittelbach, Ulrike Fischer, Chris Rowley

Written:	December, 2019
Minor updates:	September, 2020
Auto-tagged version by LaTeX:	April 2022

Contents

1   Introduction

The project described in this Evaluation Document is for the creation of an extended version of the LaTeX Document Preparation System [17, 18, 19, 22] that fully supports the production of 'Tagged PDF' [2, 15] documents in order to support different PDF standards such as PDF/UA.

One of the major advantages of such PDF documents is the superior support they offer to various types of assistive technology now available under the general heading of 'Accessibility'. Also of contemporary importance to this aim of the project are the many legal and regulatory requirements being placed on publishers and suppliers of documents to provide accessible documents, with in some cases specific requirements on the contents and coding of any PDF document that is provided. Other benefits of tagging are improvements in functionality, such as sophisticated text selection (for copy-and-paste) and data extraction.

1.1   Why LaTeX?

The importance and relevance of the LaTeX System to the production of tagged PDF derives from the most basic property of a LaTeX document: that it is a *tagged document*. More precisely, a LaTeX file is a 'free-form' text file that describes the structure of a document through markup tags, and represents its textual content as unformatted character data. A LaTeX file does not need to contain any explicit formatting information (no font information nor layout information). All such non-structural information that is needed to produce a formatted and typeset document (typically a PDF file) is normally stored separately from the document itself, in LaTeX package files and class files. Because these files are not part of a document, each can easily be used with different documents.[1] Also, a document designer or author can easily change the style of a document by selecting the class and package files to use. This formatting information in the class and package files is then accessed by LaTeX's formatter

---

[1]LaTeX allows the inclusion of explicit formatting directives in addition to the structural information, but this is not a required part of the document source.

and typesetting engine, whose function is to create fully formatted and expertly typeset output that can be printed on paper or (more frequently nowadays) consumed via a screen using a viewer application such as Acrobat Reader.

Thus a LaTeX document, since it is already 'fully tagged/structured', makes the very best starting point for the relatively straightforward production of a tagged PDF document. Adding to this the existence, world-wide, of a large number of PDF documents that were created by formatting a LaTeX document, results in a solid basis for a system with the potential of creating a large collection of 'ready-tagged' PDF documents.

1.2 Why this software development project?

A good question! Since we already have a structured/tagged form of a document in its LaTeX file, and we have excellent existing software for creating from this a formatted document as a PDF file, the *only* missing ingredient appears to be a simple tool to "move the tagging structure from one format to the other", with the LaTeX structural tagging being transformed smoothly into the PDF tagging structures — "task completed!".

Of course, when dealing with software (two systems in this case) there are bound to be some problem areas within such a task. In the case of LaTeX and PDF, these problems are non-trivial to solve. This is due to significant properties of the two systems that make the design and implementation of such a tool more complex and demanding than might be suggested by the superficial analysis outlined above.

The two most important reasons for this complexity derive from the histories of the two systems. They can be summarized thus:

- PDF uses 'content streams' and 'page objects' to represent the formatted document but these provide no natural way to represent the logical structure of the document. Thus the tagging structure and attributes have to be added (in the structure tree) without interfering with the page structure; then these must be augmented by a sophisticated system of pointers from each structure element to a page, and to marked content within the content stream of that page.

- LaTeX currently has no core support for carrying the attributed structure information into the PDF form of the document, or for associating formatted textual content with this structure.

It follows from these historical facts that our project must be underpinned by major changes to many of the fundamental document processing models and algorithms in the core of LaTeX. This substantial preliminary work will lead directly into the central concerns of the project:

- Extending the main functionality of LaTeX to preserve the attributed structural information contained in a document's source LaTeX file;

- Using this information to enhance the output PDF with the major requirements of tagged PDF, including the tree of attributed structure elements and the associations of these elements with marked content streams.

The Project Overview contains further details of the tasks involved.

1.2.1 Some history

This current lack of functionality can be traced directly back to the origins of both LaTeX itself and the underlying typesetting engines on which it is built. Both are, even now, essentially the same as when they were designed and first implemented over 30 years ago. Back then their sole purpose was the output of high-quality ink-on-paper, and so they did not need to retain the abstract structural information from the input files: there simply was no use

for it. Recall also that, at that time, for software to be widely usable it had to be small and efficient (i.e., the code had to be extremely compact and as concise as possible), therefore the functionality was typically stripped to the minimum of essentials.

Although there have been many adaptations of the LaTeX system to some aspects of onscreen documents (often provided by third-party add-ons), the document processing model and the core code have not changed significantly from those original systems. The LaTeX system has never had a major reimplementation nor any additions to its original document processing models. Thus it is, fundamentally, still confined to the basic functions, document layout and typesetting of text, for traditional printing, on paper. The underlying typesetting model in current LaTeX also lacks, for example, any concept of a graphics state and it has no graphical objects beyond rules; it is also missing many of the graphical transformations such as rotations.

Even when it is used with a typesetting engine that provides a specialized PDF backend, such as the pdfTeX or the LuaTeX engines, current LaTeX is not well suited to supporting the creation and full integration of the diverse range of elements (structure, text, graphics, etc.) that can be captured in a modern, fully-featured PDF document. Many of the tasks in the early phases of the project are therefore related to the completion of the following items of preliminary work that are needed to modernize the document processing functionality of the LaTeX kernel.

- Extension and refactoring of the handling of structural elements (commands and environments).

- Refactoring of the page-makeup machine to better support the creation of the necessary PDF objects and data.

- Provision of a new module for greatly improved handling of cross-references.

- Improvements to the processing of hyperlinks and outlines.

These preliminary tasks comprise about 25–30% of the work detailed in this document.

1.3 Scope of the project

This quote is adapted from a review of PDF 2.0:

> Today, PDF files may contain a variety of content besides flat text and graphics: these include logical structuring elements, interactive elements such as annotations and form-fields, layers, rich media (including video content) and three dimensional objects using U3D or PRC, and various other data formats.

We do not claim that this project will directly result in LaTeX being immediately used to produce the whole of this cornucopia of PDF content. It concentrates on the implementation of tagged PDF conforming to standards like PDF/UA, [11], PDF/A-2a [13] and PDF/A-3a [14].

1.4 Deliverables in this document

This document contains the following deliverables of this feasibility study (they are as defined in the Statement of Work):

- An engineering plan (overview) for getting LaTeX and the most common packages to output Tagged PDF.

- Breakdown of the key actions required to implement the engineering plan.

- Identification of the research work to be carried out as part of the engineering plan.

- Milestones identified for various phases of the plan.

2 Project Overview

This section provides details of all the engineering and research tasks that need to be carried out as part of this project. This includes specifying all necessary subtasks, documenting the relationships and dependencies between tasks, and listing the deliverables for each task. A rough project timeline is then presented in section 3, split into different phases with time estimates and the milestones to be reached at the end of each phase.

A more precise timeline depends partly on the availibility of sufficient funding. At the time of writing, funding for the initial phases of the project is available and we hope to attract further project sponsors to ensure that the project will finish successfully *and* in a reasonable time frame.

The research tasks listed in section 2.5 are to be carried out in parallel with the engineering tasks. Note that the new engineering tasks that will arise from these research activities are not included in the timeline presented here.

2.1 General Prerequisite Tasks

2.1.1 Change Strategy

A change strategy for incorporating the changes needed for tagging and accessibility into the LaTeX kernel and major packages.

Description Introducing changes to the LaTeX kernel or to core LaTeX packages is a difficult undertaking as it needs to be done without, if at all possible, breaking older documents and existing packages. In practice, any change to the core LaTeX software has the potential to break existing workflows, packages and documents. Thus these can adversely affect many parts of the LaTeX user base: both the ordinary users such as authors, editors, etc., and also the developers of classes, packages, etc.

The project goal of providing fully automated tagging support from within LaTeX requires extensive changes to the inner workings of LaTeX, so it is unlikely that such breakages can be completely avoided. It is therefore necessary to develop processes and mechanisms to

- identify such breakages;

- provide solutions that ameliorate or avoid disruptions to any part of the LaTeX user base.

This task is a *necessary* and *important* prerequisite of the overall project as LaTeX is viewed by its large community as a stable system with good backwards compatibility. Thus major disruptions will cause extremely bad publicity with the result that the adoption rate for the extensions provided by this project may be too low, and hence that the project will fail.

Once defined these actions need to be implemented as part of all the other topics/goals.

Subtasks —*none for this task*—

Dependencies and prerequisites This task is a predecessor of all other technical tasks that involve changing the code base of LaTeX. The results of this task (the defined processes and necessary actions) need to be implemented as part of all the other technical tasks.

Deliverables

- A document describing the change strategy, i.e., processes, actions and checkpoints.

- Change strategy actions to be added to all other tasks.

2.1.2 Improved test and development environment

An improved development and testing environment for LaTeX to address the specific requirements of developing and testing code related to PDF features such as structure.

Description The LaTeX Project already has processes in place that allow for automatic testing of the LaTeX code and its dependencies (regression, unit and feature tests). However, up to now these processes have tested only the direct results produced by the underlying engine, e.g., computational results, typesetting results (object placements, etc.) The correct production of a final PDF document is not currently part of the testing process because this was considered to be an automatic conversion that is always correctly done by the typesetting engine or by a post-processor.

This approach has up to now been adequate, given the current scope of LaTeX document processing: LaTeX today being concerned only with the processing of user input and with typesetting the document on (paper) pages according to the design specifications, with the final PDF being simply a representation of the spatial relationships between typeset elements (and nothing more).

When extending the scope of LaTeX to the automatic production of structured PDF, it will become important to have mechanisms and tools that automatically check whether a generated PDF conforms to its specifications and whether the structures specified in the input source have been correctly transformed into the corresponding PDF structure elements.

The purpose of this task is therefore to get or create tools that allow automated testing of PDF features, including structures, as part of a regression test suite. Also, it will develop/update the processes and scripts used by LaTeX developers to verify automatically the correct behavior of new code related to tagging and accessibility.

Subtasks

1. Identify command-line and other tools for automated testing, and develop appropriate processes and methods for their use.

2. Extend the LaTeX testing environment (`l3build` [16]) to support the testing of all PDF features, such as structure.

3. Provide sufficient test coverage in all tasks that involve coding.

Dependencies and prerequisites Each coding task needs to have a deliverable of ensuring that sufficient test coverage is provided. These are the coding tasks: 2.2.1–2.2.6 and 2.3.1–2.3.9.

Deliverables

- An extended `l3build` development and test environment that covers the testing of PDF features, including structure, etc.

- Documentation that describes how the new test processes should be applied.

2.2 General LaTeX Extension Tasks

2.2.1 Extend LaTeX core — PDF text string support

Robust tools that convert LaTeX input into valid PDF text strings.

Description   User input in LATEX documents is not always in a format that allows its direct inclusion in PDF text strings, or its use as a PDF name, as needed for alternate text, bookmarks, destination names, etc.

It is therefore necessary to provide a standard solution that reliably converts any user data (including maybe a simple formula) into the format and encoding required in PDF text string and name objects.

Subtasks

1. Review existing (incompatible) ad hoc solutions to convert user input into a PDF text string or a PDF name, and develop a standard approach that can be used by the LATEX kernel and external packages.

2. Add this standard solution to the LATEX kernel and arrange for it to replace any ad hoc solutions in the core packages.

3. Replace any ad hoc solutions in (the small set of) external packages with the new standard.

4. Document the use of the interfaces in package code.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- The change strategy being defined (2.1.1).

Deliverables

- Standardized LATEX code for conversion of user data to a PDF text string or a PDF name, replacing existing ad hoc solutions (subtask 1).

- Sufficient test coverage of the interface implementation of this new standard in the LATEX kernel and core packages (subtask 2).

- Ad hoc solutions in packages replaced by this new standard (subtask 3).

- Documentation of the interfaces of the new standard, for package developers (subtask 4).

2.2.2   Extend LATEX core — Cross-referencing generalization

A standard method to collect data generated by one part of the document processing for reuse in other parts.

Description   Cross-references in documents use data gathered in one part, such as a section or page number, to refer to that section or that page in a different part of the document, or even across different LATEX documents.

Due to its limited scope (of producing only printed documents) current LATEX has a very rigid model for this: for most cross-references it consistently passes only a 'current reference number' and a page number; for table of contents entries it always passes the 'type' of the section or object, its (textual) title or caption, and its page number.

For the production of structured PDF output this model is inadequate; it needs to be replaced by a more flexible model using property lists. This will extend the coverage of both the properties (keys for the data that is passed) and also the data (values) that can be collected and passed around. It will also need be usable with a wider range of document elements.

There already exist some ad hoc attempts to augment the current standard solution for specific purposes, but these are mutually incompatible and they also do not offer a level of abstraction suited to the necessary extensions.

Subtasks

1. Replace LaTeX's current cross-referencing mechanism with a more general version that supports the collection and passing of a wide variety of data for a range of elements and properties.

2. Add this solution to the LaTeX kernel and use it in appropriate places to pass the additional data needed for the generation of structured PDF.

3. Replace the ad hoc solutions (a small set) by this new version.

4. Document the use of this new (generalized) cross-referencing version and its interfaces in package code.

Dependencies and prerequisites

- The change strategy being defined (2.1.1).
- A suitable test environment (2.1.2).

Deliverables

- A generalized cross-reference model and implementation that supports the generation of structured PDF and extends the existing ad hoc solutions (subtask 1).
- Test coverage of the interface implementation and its use in the LaTeX kernel (subtask 2).
- Updates/extensions to existing ad hoc solutions, using this new version (subtask 3).
- Documentation of the interfaces of the new version, for package developers (subtask 4).

2.2.3 Extend LaTeX core — Hyperlinking support

Hyperlinking capabilities, including setting up destinations to which hyperlinks or outlines can point.

Description   The document outline and hyperlinks, both within documents and to external resources, are currently not provided by the LaTeX kernel itself, so they must be created through external packages such as `hyperref` [9] which need to be explicitly loaded by documents. There is a major problem with this current setup: every such external package must patch a large amount of code in order to add the necessary structures for the creation of links, outlines, etc. An important example is the specification of PDF destinations (which are called anchors in the LaTeX world), areas on a page to be used as link targets.

These patches cover both internal LaTeX kernel code and also code from many other packages, both core and external. Both the number and the varied locations of the necessary redefinitions mean that currently this patching is not very robust and can fail in several scenarios, either completely or in subtle ways: for example, by pointing to the wrong position in a document.

As outlines and links are an integral part of well-structured PDFs, it is important to replace this current variety of ad hoc alterations (to both kernel and package code) needed to support outlines and links. The replacement will be a set of well-structured kernel interfaces that provide reliable support for all the necessary operations including the specification of destinations.

The document-level interfaces provided by `hyperref` and associated packages are well established (by widespread use); they need no, or only little, adjustment as they can (with a few exceptions) be added unaltered to the LaTeX kernel. The document-level interfaces for explicitly adding outlines may need adjustments; these are handled in task 2.2.4.

Subtasks

1. Review existing ad hoc solutions to provide destinations for both outlines and hyperlinking. Develop a standard approach that can be used by the LaTeX kernel and by external packages.

2. Add the new standard programming-layer interface for destinations to the LaTeX kernel.

3. Augment all relevant standard document elements to include named destinations that are correctly positioned.

4. Add the `hyperref` document interfaces (as appropriate) to the LaTeX kernel and then retire the package itself.

5. Document how to use the user interfaces. Most of that documentation already exists as part of the `hyperref` documentation, so only minor corrections and integration into the core LaTeX documentation is needed.

6. Document how to use the programming-layer interface to add destinations to elements supported by commands and environments in external packages.

   Compile a list of all important external packages that should be updated to integrate this destination support. This list should include a rough time estimate for the work necessary and define priorities for scheduling the work if it has to be undertaken as part of the project (unfortunately, not all packages in the LaTeX world have active maintainers—even if the package is widely used).

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- LaTeX kernel support for PDF text string conversion (2.2.1).

- LaTeX kernel support for a generalized cross-reference method (2.2.2).

- Subtasks 2, 3, 4 and 6 depend on the change strategy (task 2.1.1).

Deliverables

- The standardized programming-layer interface for setting destinations (subtask 1).

- An augmented kernel that supports destinations for all relevant document-level elements (subtask 2 and 3).

- An augmented kernel that supports all user-level interfaces for hyperlinking (subtasks 4 and 5).

- Test coverage of the interface implementation and the kernel augmentation with destinations and the `hyperref` user interface (subtasks 2, 3 and 4).

- Documentation covering how to use the programming-layer interface in packages (subtask 6).

- An evaluation of which important external packages should be updated to use the programming-layer interfaces, including rough time and resource requirements for use in task 2.4.3 (subtask 6).

2.2.4 Extend LaTeX core — Outlines (bookmark) support

The document outline and its customization.

Description The document outline shares a lot of the underlying functionality with links, such as the use of destinations (task 2.2.3). However, the document-level interfaces and the customization possibilities are different, therefore outline production has been given its own task.

Subtasks

1. Review existing ad hoc solutions to provide a document outline. Design and implement a standard solution for use in the LaTeX kernel and external packages.

2. Add this solution to the LaTeX kernel and use these to replace the ad hoc solutions. Retire existing packages.

3. Document how to produce the document outline automatically. Document the interface for customizing this mechanism, and the interface for manually adding outline items.

Dependencies and prerequisites

- LaTeX kernel support for PDF text string conversion (2.2.1).

- LaTeX kernel support for generalized cross-references (2.2.2).

- LaTeX kernel support for destinations (2.2.3).

Deliverables

- Standard interface for generating the document outline (subtask 1).

- Kernel augmented to support generating outline items for all relevant document-level elements (subtask 2).

- Kernel augmented with interface for customizing the production of outline items (subtask 2).

- Test coverage of the interface and implementation, of the augmented kernel, and of the user interface for customizing the document outline (subtasks 1 and 2).

- Documentation of how to make use of the package interfaces that automatically produce outline items, and of how to manually set up outline items in documents. (subtask 3).

2.2.5 Extend LaTeX core — Hook and configuration management

A standard set of hooks in the LaTeX document processor. Well-managed hooks enable code for extensions and configuration to be executed in a reliable way.

Description The LaTeX kernel needs well-defined places where code can be reliably added by different packages without generating conflicts; these are called hooks. In particular, such hooks are needed at the beginning and the end of processing the contents of a page, just before it is used to create the content stream. Also, at the end of the document it is necessary to output some PDF objects such as the /ParentTree.

When LaTeX 2_ε was designed and implemented in the eighties and nineties, computer memory was a very limited resource and, in order to fit into the available space, many useful locations for code extension (hooks) were omitted. It is therefore necessary to change

the L^ATEX kernel software by providing the necessary hooks and configuration points; this will require an interface for managing the system, enabling kernel software and extension packages to add, in a controlled way without conflicts, code in such places.

The purpose of this task is to provide this important basic infrastructure so that code can be reliably added to produce structured PDF.

Subtasks

1. Augment the existing kernel by adding hooks and configuration points in places where code needs to be altered for the output of structured PDF.

2. Design and implement a programming-layer interface to manage adding code to hooks and configuration points.

3. Add this interface to the L^ATEX kernel, so that it can be used in other tasks (e.g., 2.2.6, 2.3.7, etc.) and by external packages that need to execute code at specific points during the document processing.

4. Document how to use the programming-layer interface to hooks in external packages.

   Compile a list of important packages that should be updated to use this new programming-layer interface. This list should include rough time and resource requirement estimates for the necessary work.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- Subtasks 1, 3 and 4 depend on the change strategy (task 2.1.1).

- This task is a prerequisite for tasks 2.2.6 and 2.3.1.

Deliverables

- Kernel augmented to contain hooks and configuration points, at least for the support of structured PDF (subtask 1).

- Standardized programming-layer interface for adding code to hooks and configuration points (subtask 2).

- Test coverage of the interface implementation, the kernel augmentation and the necessary hooks (subtasks 1 and 3).

- Documentation of the programming-layer interface in packages (subtask 4).

- An evaluation of which important external packages should be updated to use the programming-layer interfaces, including rough time and resource requirements, for use in task 2.4.3 (subtask 4).

2.2.6 Extend L^ATEX core — PDF object management

L^ATEX interfaces for managed and controlled access (by both the core and external packages) to the creation, writing out and manipulation of PDF objects. Examples of such objects are the page resources, the catalog, annotations, XObjects, etc.

Description   The current LATEX kernel can be described as basically unaware of PDF; by which we mean that, although the majority of formatted documents created with LATEX are PDFs, LATEX doesn't provide any commands or interfaces for the direct creation of PDF objects, or for writing them out to the PDF file. There is therefore currently no support for creating annotations or for adding entries to the page resources, the catalog or other major dictionaries.

Instead, all aspects of PDF generation are currently delegated by LATEX to the underlying TEX engine or to a post-processor in the workflow. In this latter case, LATEX generates a device-independent description of each page (without structural information) from which the PDF is then generated.

However, the different underlying TEX engines do all offer commands to directly write and manipulate many PDF structures at a very detailed level (or they support the passing of such information to post-processors to do this); but almost none of this functionality is used by any of the core LATEX software, which currently restricts itself to the management of the print workflow, ignoring all other aspects of PDF production.

There is some support in add-on packages for the creation of the following PDF features: links, annotations, optional content, embedding of files, some special color effects, etc. These packages use the engine-level commands in different ad hoc ways without knowledge about how other packages are attempting to write to or manipulate the same PDF structures. As a result, there are conflicts between packages (and partial overwrites) which can, depending on the situation, lead to broken PDFs.

Furthermore, as LATEX is used with a number of engines and backends that provide different sets of, sometimes equivalent, primitive commands, each package has to maintain its own set of driver files for the various engines and backends: the result is an organizational nightmare.

The main purpose of this task is therefore to develop standard interfaces to be used in a controlled way as the basis of all access to PDF objects from LATEX code. These will also have backend support and they will be part of the LATEX kernel. This will overcome the current shortcomings and allow all extension packages that manipulate PDF objects to coexist safely.

Subtasks

1. Design and implement a programming-layer interface to manage the creation, access to and updating of all types of PDF objects, including, for example, the catalog, the page resources, annotations and XObjects. Also, abstract from the peculiarities of the syntax of the different backends used by LATEX.

2. Add this interface to the LATEX kernel so that it can be used in other tasks (e.g., 2.3.1) and by external packages that want to write PDF objects (task 2.4.3).

3. Document how to use the programming-layer interfaces from the kernel in external packages.

   Compile a list of important packages that should be updated to use the new standard programming-layer interfaces. This list should include rough time and resource requirement estimates for the necessary work.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- As PDF is a page based format, some PDF objects need to be managed on a per page basis. This requires the creation of new hooks in various places in the LATEX kernel code. This makes task 2.2.5 an important prerequisite.

- Metadata management (2.3.4) must be coordinated with this task.

- Subtasks 2 and 3 depend on the change strategy (task 2.1.1).

12

Deliverables

- The kernel's programming-layer interface for accessing and manipulating PDF objects. All the related drivers in the backend, for at least these routes: `luatex`, `pdftex`, `dvips` and `(x)dvipdfmx` (subtask 1).

- Test coverage of the interface implementation and its behavior with all processes and backends (subtasks 1 and 2).

- Documentation of how to use the programming-layer interface in packages (subtask 3).

- An evaluation of which external packages should be updated to use the programming-layer interfaces, including rough time and resource requirements for use in this task 2.4.3 (subtask 3).

2.3 Structured PDF Tasks

2.3.1 Core tagging support

The code and infrastructure needed to support tagging of PDFs.

Description   To produce a structured PDF, a number of specific operators and objects have to be created and added to the PDF.

- Marked-content operators must be added to the page content streams.

- The logical structure of a document must be described by adding a hierarchy of objects called the structure hierarchy or structure tree. The marked-content operators are leaf nodes in this structure.

  Entries in the dictionaries of these objects, such as layout attributes or alternative text, can describe various aspects of the structure elements and so enhance any subsequent further processing of the PDF.

- To identify roles, attributes and associated files, and to allow PDF consumer applications to find structure elements from content items, a number of dictionaries, name and number trees have to be created, managed and referenced in the structure tree root. Examples are the number tree `/ParentTree`, and the dictionaries `/RoleMap` and `/ClassMap`.

To generate the necessary PDF objects and operators in the PDF, low-level code has to be added to the LaTeX kernel; in the case of LuaTeX engines this will be partly as Lua code.

In LaTeX's processing model the output (pages) are asynchronously generated, so the necessary data for the above operators and objects needs to be collected, managed and, during page production, added to the PDF.

Besides the kernel code, all packages that should be usable when producing structured PDF, need to be enhanced either to write the appropriate operators and objects or to provide the necessary data so that these can be written by kernel processes. This requires the design and implementation of a suitable programming-layer interface, and in a later phase the conversion of packages to use that interface.

For this task there already exists some experimental code written by the LaTeX team [21]. This code needs to be further extended and then moved from this prototype to a production-ready version.

Subtasks

1. Review the existing experimental code and add any missing parts, e.g., better interfaces to declare and use attributes and to support the `/IdTree`. The new features in PDF 2.0, such as `/PronunciationLexicon`, associated files and name spaces, will also need such additional support.

2. Verify that the code works correctly with documents written in scripts that use writing directions other than left-to-right. If necessary adjust it. This will only verify basic functionality/correctness. Full support for documents with complex writing directions is a research task (not undertaken as part of this project plan).

3. Design and implement a programming-layer interface for use by packages in the places where marked-content operators and the logical structure elements should be created when needed.

4. Design and implement a document-level interface to activate the code when a structured PDF is to be created.

5. Move the low-level code and the programming-layer interface into the LaTeX kernel so that it can be used in other tasks (e.g., 2.3.2, 2.3.3, etc.) and by external packages when generating structured PDF (task 2.4.3).

6. Document how to use the programming-layer interfaces in external packages.

   Compile a list of important packages that should be updated to use the new standard programming-layer interfaces. This list should include rough time and resource requirement estimates for the necessary work.

   The number of external packages that should be updated is unfortunately rather large in this particular instance!

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- Generalized cross-referencing support (2.2.2).

- Hook management, and page and document-level hooks are available (2.2.5).

- The PDF resource management (2.2.6).

Deliverables

- A standardized programming-layer interface to specify those places where marked-content operators and logical structure elements should be created (subtask 3).

- Document-level interface for the activation of structured PDF generation (subtask 4).

- Sufficient test coverage of the interface implementation and its correct behavior in kernel and package code (subtasks 1, 2, 3, 4 and 5).

- Documentation of how to use the programming-layer interface in packages (subtask 6).

- An evaluation of which important external packages should be updated to use the programming-layer interfaces, including rough time and resource requirements for use in task 2.4.3 (subtask 6).

2.3.2 Tagging paragraphs

The tagging of paragraphs is an essential component of structured PDFs and a requirement for tagged PDF. But doing this automatically with LaTeX is unfortunately not easy and will require a large amount of work, which is the reason why it is defined as a task on its own.

Description Paragraphs in LaTeX are normally not explicitly marked by the user, but automatically detected by the software through several mechanisms. In addition, the TeX paragraph mechanisms are also used in situations that do not involve typesetting actual 'textual paragraphs'. This makes it difficult to create correct logical structure elements for paragraphs and to add the associated marked-content operators in exactly the right places. Also, care is needed to avoid adding markup in places where this would result in spurious extra marked-content operators.

LaTeX has internally the ability to seize control at the start and the end of paragraphs, but the code executed currently in these places is very tightly linked to the low-level handling of lists, headings, etc., and it was written with the focus on compactness and speed, rather than extensibility.

Thus, the necessary augmentation of the LaTeX kernel code requires coordination across many different (and seemingly unrelated) areas of LaTeX and many external packages in order to ensure that there are no unintended side-effects or clashes.

Subtasks

1. Design and implement a mechanism for tagging text paragraphs. This requires different implementations for the different TeX engines as their abilities are noticeably different in this area.

2. Identify all places in the LaTeX kernel where the low-level paragraph mechanisms are used and ensure that they are updated to work with this implementation without producing unwanted side-effects.

3. Identify places where the code could lead to spurious markup and disable the tagging there.

4. Move the low-level code and the programming-layer interface into the LaTeX kernel.

5. Document how to use the programming-layer interface for paragraph tagging in external packages, and how to make existing code compatible with this interface.

6. Compile a list of important packages that should be updated to use the new standard programming-layer interface for paragraph tagging. This list should include rough time and resource requirement estimates for the necessary work.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- Core tagging support is available (2.3.1).

- Hook management is available (2.2.5).

Deliverables

- An implementation for automated tagging of paragraphs that is compatible with standard structures from the LaTeX kernel (subtasks 1, 2, 3 and 4).

- Sufficient test coverage of the interface implementation and its correct behavior in kernel and package code (subtasks 1, 2, 3 and 4).

- Documentation covering how to use the programming-layer interface in packages (subtask 5).

- An evaluation of which important external packages need to be updated to be compatible with this implementation, including rough time and resource requirements for use in task 2.4.3 (subtask 6).

2.3.3 Tagging basic elements

Make all standard basic LATEX structures, such as headings, lists, etc. 'tagging aware'.

Description   Almost every LATEX document uses basic structures such as sectioning commands or lists for which the project needs to provide support for tagging. In most cases this support will need to be configurable and provide additional interfaces to add, for example, alternative descriptions (task 2.3.5) or attributes and values.

For many of these basic structures there are external packages or classes that extend or alter the structure by either redefining or patching core LATEX commands. Any changes to these core commands will have a high potential for breaking existing documents and so they must be carefully planned and tested.

This also means that, besides making core LATEX structures ready for tagging, it is essential to the project's success that it addresses all necessary updates to external packages (which in this particular case is a rather large undertaking).

The main structures which should be considered in this task are:

- sectioning commands

- headers and footers of pages (which in most cases will be marked as artifact)

- numbered, bulleted and description lists: since these often contain paragraphs, the coding here must be coordinated with 2.3.2.

- table of contents, table of figures and other similar list-like structures

- float environments and captions of figures and tables

- verbatim and code typesetting

- the title or title page of the document

- footnotes

- citation commands and the bibliography

- side notes

- language changes

- other standard structures, including: quotes and quotations (inline and displayed), emphasized and emboldened text, etc.

Subtasks   For all of the structures mentioned above the following subtasks must be done:

1. Analyze which standard tagging markup the structure should provide.

2. Implement the standard tagging commands and provide interfaces to adapt them if a document has special markup needs.

3. Compile a list of all important core and external packages and classes that use, redefine or patch any particular structure.

In addition there are the following general subtasks:

1. Document how to use any special interfaces provided for the tagging-enabled basic structures.

2. Based on the outcome of subtask 3 regarding individual structures, compile a list of all important packages that should be updated so that they don't conflict with the additional tagging code added to the kernel definitions of the structures. Also, ensure that the package versions of each structure are also tagging enabled.

   This list should include rough time and resource requirement estimates for the work necessary.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- Core tagging support is available (2.3.1).

- Hook management is available (2.2.5).

- For some structures the task must be coordinated with the refactoring of the hyperlinking support (2.2.3).

- All subtasks depend on the change strategy (2.1.1).

Deliverables

- All basic standard LaTeX structures are tagging-enabled with appropriate interfaces for customization (subtasks 1 and 2).

- Sufficient test coverage of all updated commands and environments (subtask 2).

- Documentation of how to use the tagging-enabled structures (subtask 1).

- An evaluation of which important external packages should be updated to ensure that structures remain tagging-enabled when these packages are loaded, including rough time and resource requirements for use in task 2.4.3 (subtask 2).

2.3.4 Metadata management

A standard interface for specifying all document-related metadata in a uniform way.

Description   LaTeX documents typically contain a number of settings and declarations related to the document as a whole and to its processing. Examples of such settings are the values of the author, title and date, or the language (or languages) of the document. Other such information is related to the target output, e.g., which backend to use in the workflow and declarations specific to that backend, for example the PDF version or standard to use. Some of this data is used by LaTeX during typesetting, some is only passed on to the backend and some is needed in both places.

The values needed to specify XMP (eXtensible Metadata Platform [12]) data in the PDF file form an important part of the document metadata. Support for this type of metadata is currently provided in ad hoc and incompatible ways by the packages pdfx [20] (see also 2.3.9), hyperxmp [10] and in a limited way by hyperref [9]. As the availability of XMP metadata is an important requirement for several PDF standards (e.g., PDF/UA [11]), LaTeX must be extended to provide a standardized interface for it, and this interface must work in conjunction with all the code for tagging and structured PDF.

Currently this type of data is specified, in a variety of ways and places, in the document preamble: for example, through class and package options, or with commands provided by the class or by a package. This is both confusing and error prone as users may set values at too late a point in the preamble, or in conflicting ways. It is therefore necessary to define a standard interface for declaring such metadata in well-defined places in the LaTeX source.

Subtasks

1. Evaluate the methods `pdfx`, `hyperxmp` and `hyperref` provide for specifying and handling XMP metadata.

   Determine what other metadata should be managed by the LaTeX kernel, and how external packages can query and use this data.

   Based on the results of this evaluation, design and implement programming-layer interfaces for setting, and querying XMP and other metadata.

2. Design and implement a standard document-level interface for specifying the types of metadata used by LaTeX and PDF.

3. Move the programming-layer interface and the document-level interface into the LaTeX kernel.

4. Document how to use the programming-layer interfaces for metadata in external packages and how to make existing code compatible with it.

   Document also the use of the document-level interface for setting metadata.

5. Compile a list of important packages that offer options or commands to set up such metadata and so must be updated to use the new standard programming-layer interfaces. This list should include rough estimates of the time and resource requirements for the necessary work.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- PDF text string support (2.2.1).

- The task must be coordinated with the refactoring of the hyperlinking support (2.2.3).

- All subtasks depend on the change strategy (task 2.1.1).

Deliverables

- An implementation of the handling of XMP and other metadata that is compatible with the standard structures from the LaTeX kernel (subtask 1).

- Document-level interfaces to set up XMP and other metadata (subtask 2).

- Good test coverage of the interfaces and the XMP implementation (1, 2).

- Documentation covering how to use the programming-layer interfaces in packages (subtask 4).

- An evaluation of which important external packages need to be adapted to the new standard interfaces, including rough time and resource requirements for use in task 2.4.3 (subtask 5).

- Documentation of how different types of metadata should be specified in a document (subtask 4).

2.3.5 Alternate text support

The PDF format uses a number of dictionary entries to enhance a structure element or some marked-content with additional material such as alternate text. LATEX should support the addition of such alternate descriptions to a PDF document.

Description There are a few keys commonly used for the addition of such material: `/Alt` for an alternative textual description of the content of an element; `/ActualText` for the exact replacement of some symbol, word or phrase; `/E` for the expanded form of an acronym; `/PhoneticAlphabet` and `/Phoneme` for pronunciation hints (PDF 2.0).

For the first three keys there are well-known use cases, so support on the code and document level should be provided. Support for pronunciation is currently not planned as part of this project.

Alternate textual descriptions are typically added by explicit user input since they cannot, in most cases, be generated automatically. So interfaces for their input by authors are needed. This input must then be converted into valid PDF text strings.

As alternate text can be long, it is important to design the user interface in such a way that this text can be supplied using different methods, such as explicit inline input or by reading it in from an external file.

Subtasks

1. Design and implement a user interface for alternate descriptions that can be added to standard LATEX commands in a compatible way, so that existing documents continue to work.

2. Add the interface to all standard LATEX constructs where applicable. For example, commands for including graphics and LATEX environments for equations will need an interface for `/Alt`.

3. Document how to implement this user interface in external commands and environments.

   Compile a list of commands and environments in important external packages that need to offer a user interface to alternate descriptions. For example, glossary and acronym packages like `glossaries` [8] or `acro` [1] should probably have interfaces for `/E`.

   This list should include rough time and resource requirement estimates for the necessary work and define priorities for scheduling the work if it has to be undertaken as part of the project.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- LATEX kernel support for PDF text string conversion (2.2.1).

- LATEX kernel support for PDF object management (2.2.6).

- Core tagging support should be available (2.3.1).

- Subtask 1 is part of the general task to check how standard structures can be made ready for structured PDF.

- Subtasks 2 and 3 depend on the change strategy (2.1.1).

Deliverables

- The user interface for adding alternate descriptions designed, implemented and documented (subtask 1).

- Updates of all relevant standard LaTeX constructs (commands and environments) to support this interface (subtask 2).

- Sufficient test coverage of the interface implementation and its use in all standard LaTeX constructs (subtasks 1 and 2).

- Documentation covering how to implement the user interface for commands and environments in external packages (subtask 3).

- An evaluation of which important external packages need to provide a user interface for adding alternate descriptions, including rough time and resource requirements for use in task 2.4.3 (subtask 3).

2.3.6 Associated file support

Support for references to embedded or external files through `/Filespec` and the `/AF` key (introduced in PDF 2.0).

Description   The PDF 2.0 standard introduced the concept of associated files. Associated files are references from various PDF objects to external or embedded files. For example, a MathML or LaTeX source can be referenced from the structure object of an equation, a code listing can be referenced from some verbatim text, a csv file could be attached to a table, or an audio clip could describe a picture. This leads to better alternate descriptions than using the `/Alt` key, which restricts the content to the format of a PDF text string. Making use of these possibilities will need support from the PDF consumer application, but LaTeX should nevertheless now add the interfaces for this.

Workflows to *create* these useful files for embedding are not part of this task, but should be considered for later phases of the project. Examples of the content of such files include the MathML derived from the LaTeX source of an equation, or the verbatim text from some code listing.

Subtasks

1. Provide internal LaTeX support and the necessary programming-layer interfaces for adding `/AF` keys to the structure objects.

2. Review existing (incompatible) ad hoc solutions for `/FileSpec`, and develop a single standard approach that can be used by the LaTeX kernel and external packages.

   Add the solution to the LaTeX kernel, and arrange for replacing the ad hoc solutions with the new standard.

3. Design and implement a user interface for adding references to associated files. This interface can then be incorporated into standard LaTeX commands in an upward compatible way, so that existing documents continue to work.

4. Document how to implement the user interface in external commands and environments.

   Compile a list of commands and environments from important external packages that need to offer a user interface to associated files. This list should include a rough time estimate for the necessary work, and define priorities for scheduling the work if it has to be undertaken as part of the project.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- The support of `pdftex` requires an engine update that allows setting the PDF major version. This update will be available in all major TeX/LaTeX distributions from 2020 onwards.

- Subtask 2 depends on 2.2.6.

- Subtask 3 is part of the general task to check how standard structures can be made ready for structured PDF.

- Subtasks 1, 3 and 4 depend on the change strategy (task 2.1.1). They also require basic general support for structured PDF in these packages.

Deliverables

- Standardized LaTeX code supporting `/AF` and `/FileSpec`, replacing existing ad hoc solutions (subtasks 1 and 2).

- The user interface for adding references to associated files, designed and documented to the normal standards (subtask 3).

- Sufficient test coverage of the standardized code and interface implementation (subtasks 1, 2 and 3).

- Documentation covering how to implement the user interface for commands and environments in external packages (subtask 4).

- An evaluation of which important external packages need to provide a user interface for associate files, including rough time and resource requirements for use in task 2.4.3 (subtask 4).

2.3.7 Tagging table structures

Tables often contain important data needed to understand a document. A properly tagged table helps readers both to navigate the table and to extract this data.

Description   There are quite a number of environments and commands, both in LaTeX and external packages, that support the typesetting of tables. However, with one exception, they all deal only with markup for placing data into different cells and for the fine-tuning of the table layout—markup for explicitly identifying the nature of each cell is not provided.

As a consequence, automated tagging of the table content, while preserving current input syntax, will produce incompletely tagged PDFs, because the distinction between different types of table cells can only be determined heuristically.

Currently there is some level of control available for setting up column structures, but nothing comparable for marking up rows as being special, e.g., indicating that they are header rows. So even visual identification (such as boldened text throughout a row) currently must be done at the level of each individual cell, which is both cumbersome and also difficult to parse for automated tagging.

Improving this situation therefore requires a new LaTeX model for marking up table data. This needs to be explored in two directions:

- Providing additional markup commands that allow users to add necessary information about the nature of individual table cells (e.g., which cells are header cells) while otherwise still using the existing approaches.

  This is handled as part of the current engineering task.

- Designing a new syntax for specifying table data that identifies the logical parts of a table correctly while additionally allowing for fine-tuning the typographical appearance.

  This requires research and is discussed further in the research task 2.5.2.

The first approach is important for users working with existing documents from which high-quality tagged PDF should be produced. For these users, augmentation of the existing table markup with additional markup is most likely preferable to a complete rewrite of the table data in a different syntax.

The second approach is intended for new documents where an improved markup syntax will help to automatically provide correctly tagged tables.

In either case users will have to learn new syntax for specifying table data and thus an important aspect of this task is to provide sufficient documentation.

Subtasks

1. Design and implement markup commands and declarations for LaTeX's existing table syntax (`tabular` environment and `array` [3] package extension) that support the markup of table headers and table cells for tagging and for adding table attributes.

2. Move this markup interface into the LaTeX kernel and make it available in standard table packages.

3. Document how users can use the markup from subtask 1 to correctly tag tables when using the standard environments offered by LaTeX.

4. Document how to implement the basic user interface in external table packages.

   Compile a list of important external packages that should be updated to support additional markup commands for tagging tables (while otherwise preserving their current syntax). This list should include a rough time estimate for the necessary work and define priorities for scheduling the work if it has to be undertaken as part of the project.

Dependencies and prerequisites

- LaTeX kernel support for hook configuration management (2.2.5).

- Core tagging support (2.3.1).

- The change strategy (2.1.1).

Deliverables

- An implementation of additional markup commands for tagging tables that can be used with standard table markup in LaTeX (subtasks 1 and 2).

- Sufficient test coverage of this implementation and its correct behavior in the kernel and in standard packages (subtasks 1 and 2).

- User documentation and best practice guide for tagging standard LaTeX table structures (subtask 3).

- An evaluation of which important external packages need to be updated to support the new basic table markup commands, including rough time and resource requirements for use in task 2.4.3 (subtask 4).

2.3.8 Tagging mathematics

Basic support for tagging mathematical constructs.

Description   LaTeX is well known for its excellent and powerful proficiency in math type-setting, and many LaTeX documents contain elaborate mathematical content. Consequently, properly tagging such content so as to make it accessible in various ways (e.g., copy-and-paste, data extraction, etc.) is an important aspect of this project.

However, it is unfortunately not well understood how best to tag mathematical content in a PDF so that it can be usefully accessed for varied purposes. This area therefore requires substantial research (outlined in task 2.5.1) prior to defining appropriate engineering tasks.

The current engineering task is therefore limited to the provision of basic tagging. It also covers the provision of the code needed to read the mathematical content verbatim (grab it) and store it for various types of post-processing. Both of these tasks will be useful regardless of any future work defined as the outcome of the research activities.

Subtasks

1. Add to the standard environments for displayed equations some code that grabs the content of the environment verbatim so that it can, for example, be added to the `/Alt` key or saved to a file (possibly following some limited post-processing that produces other types of 'math-as-text' representation).

2. Do the same for inline math formulas, e.g., `$...$` syntax.

3. Review the LaTeX core and important external packages to find all uses of TeX's math mode purely to obtain visual effects, such as superscript characters or vertical centering. Since such uses could conflict with the additional processing code needed for actual mathematical material, they will need special treatment or recoding.

4. Add to the standard math structures the code needed to tag them as Formula elements.

5. Design and implement a user configurable method to get (possibly post-processed) mathematical content automatically added as alternate text when explicit alternate text is not supplied in the document.

6. Implement this method for all the mathematical structures of standard LaTeX.

7. Document how to implement these mechanisms for use in external math packages.

   Compile a list of important external packages that should be updated to support the new mechanisms for tagging and grabbing mathematical content. This list should include a rough time estimate for the necessary work, and define priorities for scheduling the work if it has to be undertaken as part of the project.

Dependencies and prerequisites

1. The change strategy being defined (2.1.1).

2. A suitable test environment (2.1.2).

3. Core tagging support is available (2.3.1).

4. Hook management is available (2.2.5).

5. LaTeX kernel support for PDF text string conversion (2.2.1).

6. This task needs coordination with the tasks for alternate text support (2.3.5) and associated files (2.3.6).

Deliverables

- All standard mathematical environments and inline math structures can be tagged and will be enabled to grab their content verbatim. There will also be support for limited post-processing to produce textual content for the /Alt key, or to save the data to a file (subtasks 1, 2 and 4).

- Sufficient test coverage of the implementation and its correct behavior in the kernel and standard packages (subtasks 1, 2 and 4).

- User documentation for customization of the math tagging (subtask 5).

- An evaluation of which important external packages need to be updated to support the new mechanisms for tagging and grabbing mathematical content, including rough time and resource requirements for use in task 2.4.3 (subtask 7).

2.3.9 Standards compliance: PDF/UA, PDF/X, PDF/A and possibly others

An interface and tools for creating documents that conform to specific PDF standards, when feasible.

Description   The major goal of this project is the production of structured PDF which is a prerequisite for a number of different PDF standards (PDF/UA, PDF/X, PDF/A, PDF/E, PDF/VT, etc., including different versions and conformance levels).

In most cases full compliance will not be achievable without further workflow steps, but the purpose of this task is to evaluate to what extent conformance to such standards can be automatically achieved, and to provide a document-level interface in which conformance to a certain standard can be requested, directing LaTeX to use the provided tools to make this happen.

In most cases full compliance will not be achievable without further workflow steps, but LaTeX should be enabled to provide a solid basis, to warn about possible conflicts (e.g., when unsuitable constructs are used) and to tell the user about the additional (probably manual) actions needed to achieve compliance.

In this area there is already an important package, pdfx [20] by Ross Moore et al., whose intention is to meet the requirements of the different PDF standards by providing, at least, conforming metadata and color profiles.

Part of Ross's work will be incorporated into the Metadata support task 2.3.4. This task will include an evaluation of the remainder of his work, in particular the interface for requesting standards compliance when processing a document.

Due to the fact that the different TeX typesetting engines offer slightly different features, compliance with a certain PDF standard may require the use of a specific engine (and/or additional workflow steps). One subtask is therefore to determine and document these differences and to guide users in their selection of an engine and workflow, based on the targeted result.

Subtasks

1. Analyze the work done in the package pdfx with respect to achieving compliance with different PDF standards.

2. Design and implement a document-level interface for requesting compliance.

3. Incorporate this interface and the necessary tools into the LaTeX kernel, or put them in a core package (i.e., a standard add-on package managed by the LaTeX project). This will offer the functionality of pdfx in a way that works well with the new approach to structured PDF in standard LaTeX.

4. Document to what extent the different PDF standards can be achieved automatically when using each of the available TeX engines, and what will remain to be done, probably by manual actions, in a workflow.

5. Coordinate updates to the `pdfx` package to use the kernel functionality provided by this project.

Dependencies and prerequisites

- A suitable test environment (2.1.2).

- Core tagging support is available (2.3.1, 2.3.2, and 2.3.3). This task can be worked on without extended tagging support (for tables, math, etc.) in place, but eventually this will also be needed (2.3.7, 2.3.8).

- Alternate text support is available (2.3.1).

- This task needs coordination with the Metadata management task (2.3.4).

- Subtask 3 depends on the change strategy (2.1.1).

Deliverables

- The user interface, with documentation, for requesting the production of a document to a specific PDF standard (subtask 2).

- Sufficient test coverage of the interface implementation (subtask 3).

- Documentation of how, and to what extent, different PDF standards can be achieved when using each of the TeX engines: `pdftex`, `xetex`, `luatex` (and the variants used for Asian languages) (subtask 4).

2.4   Aggregation tasks

The tasks in this section describe activities that are carried out typically as subtasks of other activities described earlier. Here we provide some extra detail.

2.4.1   User and developer acceptance testing

A testing infrastructure for extensive developer and user acceptance tests, including feedback loops to improve the project's outputs.

Description   LaTeX documents typically load a wide variety of external packages and classes, i.e., code that is not developed or managed by the LaTeX project team. A number of those have become de facto standards and are expected to be supplied with any LaTeX installation or to be obtainable from CTAN (the Comprehensive TeX Archive Network [6]). Many other packages are also available on CTAN (and often also in distributions) but these are used only occasionally. These days the CTAN catalogue lists more than 5000 packages.

Moreover, there are also a large number of packages and classes that are not so generally available, e.g., in-house styles, special publisher or university class files, packages developed by individuals for their own use, etc.

This is one reason why even the very comprehensive set of tests that is planned for all coding tasks of this project will not be able to cover a representative sample of LaTeX documents.

Furthermore, extending LaTeX to produce structured PDF will introduce new aspects, such as accessibility, for which the expert knowledge needed for testing lies, at least partially, outside the LaTeX community.

It is therefore very important to ensure that extensive user and developer acceptance testing is undertaken, preferably including testers with specific domain knowledge.

The purpose of these acceptance tests is to obtain early feedback on

- document breakages, and errors in packages or classes;

- missing or broken features;

- problematic interfaces;

- suggestions about possible enhancements.

One necessary prerequisite for extensive user and developer testing has been already provided by the LaTeX project team: as of 2019 it is possible for the team to distribute, automatically as part of the major LaTeX distributions [7], pre-releases of LaTeX that contain new or altered functionality. This makes it possible to get feedback not only from package developers or highly skilled users, but also from 'normal' users. This works well because becoming a tester no longer requires any specialized installation steps and testing can be done on the fly using private documents.

Starting from phase II of the project, testers may need tools to validate the results (e.g., correct tagging of PDFs) without the need for manual inspection.

Subtasks

1. Recruit the widest possible variety of testers, from package developers, authors, editors, publishers, companies etc., to test their existing documents for possible breakages and other problems. This will be done both by direct contact and also by publishing, through suitable channels, information about the project, its progress, and how to get involved as a tester.

   A large number of real documents, often quite complex ones, can be found on sites such as arXiv.org [4]. Sites like these should be contacted to ascertain whether automated testing of their corpora can be arranged.

2. Recruit the widest possible variety of package developers, authors, editors, publishers, companies etc. who are willing to adapt their documents to the new syntax requirements required to get a tagged PDF, either by direct contact or via publicity for the project and its progress through suitable channels.

3. Recruit testers familiar with the use of assistive technology such as screen readers.

4. Compile a list of (online and offline) software tools for validation of PDF documents, and document how to use them.

5. Setup within the project a feedback and issue tracker system specific to this type of acceptance testing.

6. For every phase of the project, it should be known what requirements a document must fulfill to allow the activation of tagging. This requires full documentation.

7. Feedback concerning external packages should be forwarded to their maintainers, preferably together with suggestions on how to resolve any identified problems: see also task 2.4.3.

Dependencies and prerequisites   This task needs to be coordinated with the creation of best practice guides and other publications (2.4.2) and with the update of important external packages (2.4.3).

Deliverables

- Recruited testers with necessary skills for testing out the software and the tools developed in each task. Communication methods established to inform them when software is ready for testing. A sufficient body of test documents has been collected and methods for testing them established (subtasks 1, 2 and 3).

- The list of tools for validation is compiled and access to a suitable set of tools is available (subtask 4).

- Issue tracking mechanisms and methods to forward feedback to external package developers are set up and advertised (subtasks 5 and 7).

- The necessary documentation to enable testing and feedback are provided. This is a deliverable of each individual task (subtask 6).

2.4.2 Best practice guides and other publications

An important aspect of this project (as with every project) is good documentation at all relevant levels.

Description   To support the 'code maintainers', the code from every task will be documented using the methods, and to the highest standards, that are common within the LaTeX community. This will be of great help to those who will in the future maintain and enhance it.

For 'class/package developers' there will be documentation and best practice guides on at least the following:

- how to add tagging support to package code, and what to avoid;

- which packages are tagging-aware and can be used;

- a discussion of problematic types of structures and commands;

- general best practice advice;

- how to validate class/package code.

'LaTeX users' (authors, editors, etc.) will need full documentation and best practice guides on

- how to activate tagging;

- how to make document elements, user definitions, etc., 'tagging aware';

- how to use the interfaces to PDF facilities such as /Alt;

- considerations necessary when producing documents conforming to a specific PDF standard, e.g., PDF/UA;

- which packages/classes can be used to generate structured PDF;

- a discussion of problematic structures and commands;

- general best practice advice;

- how to validate the final PDF.

Over time the project will enable the use of more and more (specialized) document elements for use in structured PDFs. Additionally, the set of external packages that can be used will grow. It is therefore important to provide a simple method for users to quickly check that they are using only constructs and packages that have been updated to support tagging, etc. The necessary data for making such checks should be stored in a simple database that supports easy updating over the lifetime of the project. Thus, whenever new functionality/packages become available, this can be immediately reflected in this database and other documentation.

27

There will also be a need to publish reports on the progress of the project, advertising newly available features. This will popularize the project in the community and also recruit volunteers to help with testing (and possibly with other work, such as updating packages).

Subtasks

1. Ensure that code is always fully documented.

2. Provide best practice guides for package developers.

3. Provide best practice guides for users.

4. Provide a package (`structured-pdf-check`) to verify that only supported packages and constructs are being used, with warnings otherwise.

5. Publish progress reports as needed (matching release cycles).

Dependencies and prerequisites

- For code maintainer documentation: the change strategy (2.1.1) and the improved testing system (2.1.2).

- For package writer documentation: the generalized cross-reference mechanism (2.2.2), hyperlinking support (2.2.3), hook and configuration management (2.2.5), PDF object management (2.2.6), the basic tagging interface (2.3.1), mathematics (2.3.8), and tables (2.3.7).

- For user documentation: the interfaces for hyperlinking (2.2.3), alternate text (2.3.5), associated files (2.3.6), metadata (2.3.4), mathematics (2.3.8), and tables (2.3.7).

Deliverables

- All code to be fully documented (subtask 1).

- A best practice package/class developer guide (subtask 2).

- A best practice user guide (subtask 3).

- A LaTeX package for checking that only constructs and packages supporting structured PDF generation are used in a document (subtask 4).

- Progress reports published in conjunction with software releases (subtask 5).

2.4.3   Coordination of the update of important external packages

LaTeX documents typically use a number of external packages in addition to the core LaTeX software. The success of this project requires that a substantial number of these packages get upgraded to support structured PDF.

Description   To successfully generate structured PDFs from many LaTeX documents, it is not enough to update only the LaTeX kernel and the core packages. Nearly all documents use external packages, so it will also be necessary to provide enhanced/updated versions of those packages, starting with the most important ones. From the viewpoint of this project, these external packages can be divided into the following groups:

1. Packages of high importance (i.e., used in a substantial number of documents)

    (a) with active maintainers;
    (b) without any active maintainer.

2. Specialized packages used in only few documents.

While it is not possible to give precise criteria for this classification of packages into the groups 1 and 2, a rough division is easily done and will suffice for this project.

For a successful project, all the packages in group 1 must be adapted early on (i.e., as part of the project) to produce structured PDF. Without these adaptations, it will be impossible to generate correctly structured PDF from too high a proportion of 'real documents'. Adjusting the code of packages in group 2 can be deferred to a later stage, or maybe not done at all.

A major problem here is the subgroup 1b. Many of the external packages for LaTeX were originally developed by volunteers who at some point moved on, leaving their work without a maintainer. In many cases this has not been an issue up to now, because the packages worked well and did not require active maintenance. However, for most packages the requirement to produce structured PDF will make updating unavoidable. Therefore it will be necessary either to find new volunteer maintainers within the LaTeX community or for this project itself to take on the necessary updating tasks.

Because it is not clear at this stage whether it will be possible to find new maintainers for abandoned but important packages, the amount of work necessary to be undertaken as part of the project can not be reliably estimated right now. However, it is likely that, for most packages in this category, the updating work will need to use project resources and thus lengthen the project timeline.

Subtasks

1. Provide a best practice guide (compiled from the results of various prerequisite tasks) covering how to update an existing package to support structured PDF generation.

2. Identify all external packages belonging to groups 1a and 1b and calculate the resource requirements of updating these packages.

3. Develop a plan for the order in which the packages should be updated.

4. Develop and implement a reporting and tracking method for the progress in converting external packages to support structured PDF generation. This is needed because the conversion of these packages will have to be spread over a long time period and it will involve a substantial number of developers from outside the core project workers.

5. For group 1a, coordinate with their maintainers the conversion to using the new interfaces.

6. For group 1b, make an attempt to find new maintainers. If unsuccessful, carry out the conversion to the new interfaces as part of this project.

Dependencies and prerequisites

- Input for this task will come from the following: 2.2.1(3), 2.2.2(3), 2.2.3(6), 2.2.5(4), 2.2.6(3), 2.3.1(6), 2.3.5(3), 2.3.6(4), 2.3.8(7) and 2.3.9(5).

Deliverables

- Best practice documentation for the conversion of external packages (subtask 1).

- An evaluation of the resources needed to update the large collection of external packages so that they support the production of accessible, tagged PDF. This will include resources for the coordination of the project and technical support for package writers and maintainers (subtask 2).

- A plan for doing the conversion, including progress tracking (subtasks 3 and 4).

- Updated versions of all important packages (subtasks 5 and 6).

2.5 Necessary Research Work

In contrast to the fairly well-defined engineering tasks outlined in sections 2.1 to 2.4, there are some project tasks that require more extensive research. This research is likely to identify further engineering tasks.

Three areas of necessary research are described in this section. This research work will be carried out in parallel with the engineering tasks. It is important to recognize that this research will very probably lead to additional engineering tasks, and that these will need to be clearly defined and then incorporated into the project schedule (section 3), resulting in a timeline for the project that is longer than the current estimate.

2.5.1 Research — Tagging mathematics

Description   The PDF reference offers only the type 'Formula'. as a standard structure element for equations and other mathematical material. Neither the PDF reference nor the best practice guide make any suggestions about how to sensibly markup the content of such an equation so that it becomes accessible to any PDF consumers. We are investigating the following options to overcome this:

1. Some representation of the math formula could be placed as the value of the `/Alt` key of the structure. This version must be a PDF text string, so some possibilities are: the LaTeX source for the formula; a textual (but non-verbal) version of the math using the full Unicode range of mathematical symbols and math alphabets; the text for a 'natural spoken language' version of the math.

   A light version of this will be implemented as part of engineering task 2.3.8.

2. The content of the formula could be tagged (as structured PDF (2.0) [15]) using elements from the MathML namespace [5].

3. The LaTeX source, a plain text version and/or a MathML representation of the formula could be added as associated files or perhaps even in an attribute object dictionary for user properties associated with the formula.

Options 2 and 3 both require additional engineering to allow for extensive post-processing of the mathematical content in the LaTeX source.

Not much is currently known about how PDF consumer applications can make use of such structures and of enhanced alternate descriptions, nor about which of them (or which combination) will offer the best user experience. Nor do we know much about what relevant APIs (maybe not publicly documented) are now built into Adobe DC, or can easily be engineered. Thus any implementation will depend on further research in collaboration with Adobe engineers and other experts on accessibility of math, together with input from users and the providers of PDF consumer applications.

Subtasks   (naturally incomplete as this is research)

1. Research on whether a MathML representation of the equation should be preferably added with variant 2 or 3, and consider how a suitable workflow could be implemented.

2. Explore other possible avenues to improve the use of mathematical content by PDF consumer applications.

Deliverables

- Define appropriate engineering tasks as the result of the research and adjust the project schedule as necessary.

2.5.2 Research — Tagging tables

Description   As explained in engineering task 2.3.7, the current LATEX model for specifying the structure of tables and the data they contain is not a good match to the model used by the standard PDF structure elements and attributes for tables. Thus it is difficult to automate the process of PDF tagging for table structures. Neither is the LATEX model well-suited to tasks such as data entry or editing in tables.

It is therefore necessary to research alternative models for the LATEX input that better support these two activities.

Another deficiency of the current LATEX model is its lack of support for adding markup based on the rows, rather than the columns.

Subtasks   (naturally incomplete as this is research)

1. Research models to improve data entry and manipulation in tables.

2. Review how existing core and external packages handle tables, and develop a design and plan for an improved implementation of tabular material in LATEX. This should approach table markup from a logical (rather than visual) perspective, treating rows and columns equally. But it must still offer sufficient flexibility for fine-tuning the visual presentation, of both rows and columns.

Deliverables

- A refactoring plan for replacing the existing table specification methods with a new standard interface based a new model (subtasks 1 and 2).

- Define other appropriate engineering tasks as the result of the research and adjust the project timeline as necessary.

2.5.3 Research — Using attributes

Description   Attribute objects can be used to attach additional information to any structure element. Generic support for such attributes is part of the tasks 2.3.1 and 2.3.3, and table attributes will be handled in task 2.3.7.

But to decide whether more specific interfaces and additional support is needed for the different attribute types, research is required. This should cover all types, i.e, layout attributes, list attributes, table attributes, attributes governing translation to other formats like XML or HTML, user properties and non-standard attributes.

Subtasks   (naturally incomplete as this is research)

1. Research how current PDF consumers use attributes and which of the different types and to what extent they are commonly supported in different applications.

2. Research to what extent the layout properties of a LATEX document can be automatically mapped to layout attributes.

3. Research to what extent other attribute types can be automatically mapped from information available to LATEX.

4. Determine if there is a need to provide document-level interfaces to set attributes manually, or if it is sufficient to rely on automation.

Deliverables

- Define appropriate engineering tasks as the result of the research and adjust the project schedule as necessary.

3 Project Timeline

The project is divided into six phases which follow the typical LATEX release cycles, i.e., each phase ends with a normal LATEX maintenance release. These releases normally happen in spring (between April/May) and in fall (October/November), so typically in half year intervals.

The work here is organized so that useful intermediate results will become available as soon as possible in order to attract timely feedback and early adoption. We expect, for example, that after phase 2 it will already be possible to automatically generate tagged PDFs for a restricted set of documents. In later phases, improvements will appear that extend the coverage. Also, some parts of the implementation will be moved into the core of LATEX.

Of course, during the various phases much will be 'work in progress' and so we expect testers and early adopters to work with understanding of such temporary limitations and of the possibility that extra installation steps, etc., will be necessary.

In Figure 1 on the following page the dependencies for the major tasks of the project are visualized, with the entry 'P' indicating that task *A* (on the left) is a prerequisite of task *B* (on the top). While these naturally define a partial order, there is some flexibility in ordering the tasks as explained above. Necessary research (see section 2.5) will be carried out in parallel with the engineering tasks to be executed in the six phases. Additional engineering tasks that result from this research are not accounted for in the timeline. These will have to be added as and when appropriate; they are likely to lengthen the project.

The time estimates for the indivdual tasks below assume that it is possible that at least one developer works exclusively and full time on the task during this time with support by further developers as necessary. Whether or not this will be possible in all cases will depend on various factors, but one important one is the availability of appropriate funding to free the responsible developers from undertaking other work to earn a living.

A realistic scenario would be that each phase takes one to two release cycles, which means that the overall project will stretch across four years as a minimum, but probably somewhat more. Additional funding will help to ensure timely delivery of the phase results and may allow the scope to be broadened in some areas, but any expectation of earlier delivery would not be realistic given the complexity of the topic.

It is also important to note that all updates to important external packages are to be done using external resources, i.e., by the maintainers of those packages. This assumption is probably not valid in all cases (see discussion in task 2.4.3). In that case additional work has to be undertaken as part of the project, and this will also alter the timeline.

3.1 Phase I — Prepare the ground

This phase started in February 2020 and will probably end with the spring release in 2021. The main goal of phase I is to provide the necessary basis for all the following phases.

An appropriate change strategy must be defined for all coding activities concerning the LATEX kernel. This is because this project alters the core code in significant ways, way beyond anything that happened in recent decade(s).

Similarly, an extension to the test and development environment is needed for nearly all later tasks, to support the testing of code that involves adding objects to the final PDF output.

The other tasks tackled in this phase are for adding to the LATEX kernel the functionality that is needed to implement tagging of PDF documents.

3.1.1 Tasks of phase I

Task 2.1.1 Define a change strategy for altering LATEX without causing serious issues for the world-wide user base.

Estimated time: 4 weeks

	Change Strategy	Test Env.	PDF strings	X-Refs	Hyperlinking	Outlines	Hooks	PDF object mngt	Core Tagging (Infrastr.)	Paragraph Tagging	Basic Element Tagging	Table Tagging	Math Tagging	Metadata	Alternate Text	Associated Files	PDF Standards	Documentation & Guides	Third-party Updates	User Acceptance Testing
Change Strategy		P	P	P	P	P	P	P	P	P	P	P	P	P	P	P			P	
Test Environment		P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P			D
PDF strings				P	P			P						P	P			P	P	D
X-Refs			P					P										P	P	X
Hooks						P		P	P	P	P							P	P	D
PDF object management								P							P	P		P	P	D
Hyperlinking						P			P									P	P	X
Outlines																		P		X
Core Tagging (Infrastr.)									P	P	P	P							P	↑
Paragraph Tagging									P									P		X
Basic Element Tagging																		P		↓
Table Tagging																		P		X
Math Tagging																		P		X
Metadata																P	P			X
Alternate Text																		P	P	X
Associated Files																		P	P	X
PDF Standards																		P	P	X
Documentation & Guides																			P	P
Third-party Updates																				X
User Acceptance Testing																				

P = prerequisite, X = user testing needed, D = developer testing needed

Figure 1: Dependency matrix of the project

Task 2.1.2 Improve the current test and development environment to support the development and testing of structured PDFs.

Estimated time: 6 weeks

Task 2.2.1 Implement PDF text string conversion.

Estimated time: 6 weeks

Task 2.2.5 Design and implement a hook management system for the LaTeX kernel and add hooks in all places where they are needed, for the extensions developed in later phases.

Estimated time: 10 weeks

Task 2.2.6 Design and implement programming interfaces for managing PDF objects as needed by several other tasks in later phases.

Estimated time: 10 weeks

Task 2.4.1 Start to provide developer acceptance tests for all of the tasks above. Testing for 2.1.2 and 2.2.1 should conclude in phase 1 while testing for 2.2.5 and 2.2.6 will continue in phase 2.

Estimated time for testing (external work):

- test environment: 8 weeks
- PDF string conversion: 8 weeks
- hook management: 12 weeks
- managing PDF objects: 6 weeks

Task 2.4.3 Coordinate updates to external packages with respect to PDF text strings (i.e., the functionality provided with 2.2.1).

Estimated time for updating external packages (assumed[2] to be external work):

- PDF string conversion: 8 weeks

3.1.2 Milestones of phase I

Detailed deliverables are given in the task descriptions in section 2. In contrast to later phases, this initial phase does not have any results visible to end-users except for the hook management (2.2.5) that also offers some user-level improvements. The highlights are:

- Change strategy is defined and documented;

- Development and test environment is extended (and usage documented);

- PDF text string conversions are available in the kernel;

- Standard hook management is designed, implemented and used by the kernel;

- Interfaces for PDF object management are designed and implemented (implemented as a prototype add-on package at this stage).

3.2 Phase II — Provide tagging of simple documents

This phase will probably start in spring 2021. The main goal of phase II is to provide automatic tagging of simple documents, excluding more complicated structures such as mathematics, tables, etc.

This is achieved by setting up the necessary core code that provides the general mechanisms, dealing with the issues around automatic detection of paragraph text and tagging it, and by enabling a subset of the document elements to produce tags.

3.2.1 Tasks of phase II

Task 2.3.1 Design and implement the internal code necessary for the production of tagged PDF. This task depends on the existence of the extended cross-reference mechanism (task 2.2.2), which is in Phase III, therefore it will initially use some workarounds that will be replaced in phase III.

Estimated time: 6 weeks

Task 2.3.2 Design and implement a mechanism for the automatic identification and tagging of paragraph text.

Estimated time: 6 weeks

Task 2.3.3 Implement tagging for basic document elements provided by the LaTeX kernel and core classes and packages, e.g., section headings, lists, inner paragraph elements, etc. More and more elements will become usable in tagged PDFs over time. This task will continue and finish in phase III.

Estimated time: 8 weeks (plus 4 more weeks later).

Task 2.4.1 Continue with developer acceptance testing for tasks 2.2.5 and 2.2.6 and start developer acceptance testing for 2.3.1 and 2.3.2.

Estimated time for testing (external work):

- tagged PDF support code: 8 weeks

---

[2]See the discussion in 2.4.3 on page 28 concerning this assumption.

- tagging paragraphs: 8 weeks

Task 2.4.3 Coordinate updates to external packages with respect to hook management and PDF object management (i.e., the functionality provided with tasks 2.2.5 and 2.2.6 in phase I).

Estimated time updating external packages (assumed to be external work):

- hook management: 20 weeks
- PDF object management: 12 weeks

3.2.2 Milestones of phase II

The main result of this phase will be the ability to generate usefully tagged PDFs from LaTeX documents that use only a restricted set of document elements. As task 2.3.3 is split between this and the next phase, some of the elements that are to be enabled for tagging by this task be still be out of scope at this point. Detailed deliverables are given in the individual task descriptions in section 2. The highlights are:

- Availability of the low-level mechanisms needed for tagging.

- Automatic tagging of paragraph text.

- A subset of the standard document elements is "tagging enabled".

Thus, generation of tagged PDFs for documents using a restricted set of document elements is possible, albeit at this point only through loading an additional package (provided by the project) as not all code will have yet been moved into the LaTeX kernel.

This means that it will be possible to start user acceptance testing to obtain initial feedback that will then be incorporated into later phases.

3.3 Phase III — Remove the workarounds needed for tagging

The main goal of phase III is to extend the coverage of automatic tagging and to remove the workarounds that had been necessary initially to provide a working prototype. More complex structures are still not enabled for automatic tagging.

3.3.1 Tasks of phase III

Task 2.3.3 Continue with making the remaining basic document elements "tagging aware".

Estimated time: 4 weeks

Task 2.2.2 Design and implement the extended cross-reference mechanism for LaTeX that is needed for automatic tagging. Update the LaTeX kernel to use the new mechanism.

Estimated time: 12 weeks

Task 2.3.4 Provide an interface for specifying all types of document metadata, and integrate it with the kernel.

Estimated time: 6 weeks

Task 2.3.1 Update the internal code for tagging to use the extended cross-referencing functionality, and remove the workarounds initially necessary.

Estimated time: 6 weeks

Task 2.4.1 Start with user acceptance tests (as explained in 2.4.1) for tasks 2.4.3 and 2.3.4 as soon as they are integrated into the development kernel and a pre-release is made available. Also start user acceptance testing for basic tagging within the restrictions of task 2.3.3.

Estimated time for testing (external work):

- extended cross-references: 8 weeks
- metadata: 8 weeks
- basic tagging (user-level): 12 weeks

Task 2.4.3 (After the acceptance tests have been completed) Coordinate updates to external packages with respect to the extended cross-reference mechanism (i.e., the functionality provided by 2.2.2). Estimated time updating external packages (assumed to be external work):

- cross-references: 12 weeks

3.3.2 Milestones of phase III

At the end of this phase, functionality for basic tagging will be in a state ready for inclusion into the LaTeX kernel. This will then happen in phase IV. Detailed deliverables for all the tasks are given in the individual task descriptions in section 2. The highlights are:

- An interface for specifying metadata is available and integrated in the LaTeX kernel.
- An extended cross-referencing mechanism is available and integrated in the LaTeX kernel.
- Automated tagging of documents with a restricted set of document elements is available and ready for integration into the LaTeX kernel.

3.4 Phase IV — Make basic tagging and hyperlinking available

Basic automatic tagging is now ready for inclusion into the LaTeX kernel, so the main goal of phase IV is to incorporate all the code currently in a prototype package into the kernel itself. This needs to be done carefully and cautiously as it should not have any negative impact for users processing legacy documents. The second main task of this phase is to provide support for hyperlinking in the LaTeX kernel.

3.4.1 Tasks of phase IV

Tasks 2.2.6, 2.3.1, 2.2.1 and 2.3.3 Finish off the tasks by incorporating the code in the LaTeX kernel so that it is available by default.

Estimated time: 8 weeks

Task 2.2.3 Design and implement hyperlinking and move it into the LaTeX kernel

Estimated time: 12 weeks

Task 2.3.8 Provide support for tagging the standard elements of LaTeX that contain mathematical material. See the task description for the limitations on the scope of this task. The results of this task are used in phase V and only then integrated into the LaTeX kernel.

A related research task with extended scope (not included in the time estimation) is described in 2.5.1.

Estimated time: 4 weeks

Task 2.4.1 Further user acceptance testing of tagging, which is now provided as part of the LaTeX kernel and no longer only as a prototype. Also do user acceptance testing on hyperlinking, so that both can be activated at the end of the phase.

Estimated time for testing (external work):

- basic tagging integrated (user-level): 12 weeks
- hyperlinking: 8 weeks

Task 2.4.3 Coordinate updates to external packages to make their document elements "tagging enabled", as the core infrastructure for this is now available as part of the LaTeX kernel.

Estimated time updating external packages (assumed to be external work):

- enable tagging: 80 weeks

This is a huge task and may influence the available time in later phases if some of this work has to be done as part of the project, see comments in task 2.4.3 on page 28.

3.4.2 Milestones of phase IV

At the end of this phase, the functionality for basic tagging and hyperlinking will be directly offered by the LaTeX kernel. Detailed deliverables for all tasks are given in the individual task descriptions in section 2.

3.5 Phase V — Provide extended tagging capabilities

With basic tagging now available the focus of this phase lies in providing extended support for tagging by adding tables and formulas to the supported elements. Furthermore, interfaces for specifying alternate text are being developed and added to all relevant elements, such as graphical elements.

3.5.1 Tasks of phase V

Task 2.3.7 Design and implement a markup interface that supports enhanced tagging of existing standard LaTeX table structures.

Estimated time: 6 weeks

Task 2.5.2 Design (but not implement) a new table model that allows for a better, clearer markup of table content, thereby enabling auto-tagging while preserving the current flexible control of the visual representation. This requires some up-front research, as described in the research task 2.5.2.

The time needed for implementation is not included in the estimates as it depends on the outcome of the research task and of this design activity.

Estimated time: 12 weeks

Task 2.3.5 Design and implement interfaces for providing alternate text and update relevant LaTeX environments and commands to provide this interface.

Estimated time: 8 weeks

Tasks 2.3.7 and 2.3.8 Move the code from the two tasks (after user acceptance testing) into the kernel (so that it is available in the next release). Note that the code for task 2.3.5 (Alternate text) will not be ready in time.

Estimated time: 4 weeks

Task 2.4.1 Do user acceptance testing for extended tagging (2.3.7 and 2.3.8 already prepared in phase IV) and alternate text (2.3.5).

Estimated time for testing (external work):

- extended tagging (user-level): 12 weeks
- alternate text: 8 weeks

Task 2.4.3 Continue to coordinate updates to external packages to make their document elements "tagging enabled". Start coordinating package updates with respect to hyperlinking usage, which is now offered by the kernel.

Estimated time updating external packages (assumed to be external work):

- hyperlinking: 20 weeks

3.5.2   Milestones of phase V

At the end of this phase, the functionality for extended tagging will be directly offered by the LaTeX kernel. Detailed deliverables for all tasks are in the individual task descriptions in section 2.

3.6   Phase VI — Handle standards

The goals of this phase are to provide support for the relevant PDF standards (as far as this is possible using LaTeX without post-processing the resulting PDF), and adding kernel support for outlines and associated files.

3.6.1   Tasks of phase VI

Task 2.3.9 Evaluate and document to what extent conformance to different PDF standards is possible with current LaTeX. Provide an interface for requesting that documents are formatted to conform to a specific standard (generating appropriate warnings if unsuitable elements are found in the document).

Estimated time: 8 weeks

Task 2.2.4 Design and implement support for producing customizable outlines related to the document structure, both automatically from the document's heading structure as well as manually through dedicated directives in the LaTeX source.

Estimated time: 4 weeks

Task 2.3.6 Design and implement an interface for handling associated files and update all relevant standard commands and environments of LaTeX to support this.

Estimated time: 6 weeks

Task 2.4.1 Do user acceptance testing for the above tasks so that this support can be included in the LaTeX by the end of the current phase.

Estimated time for testing (external work):

- conforming to standards: 4 weeks
- outlines: 4 weeks
- associated files: 4 weeks

Task 2.4.3 Continue coordinating updates to external packages to make their document elements "tagging enabled". Start coordinating package updates with respect to supporting the alternate text and the associated files interfaces which are now offered by the kernel.

Estimated time updating external packages (assumed to be external work):

- alternate text: 12 weeks
- associated files: 12 weeks

3.6.2   Milestones of phase VI

With the LaTeX release concluding this phase the project, as currently planned, will finish with the following milestones achieved:

- The production of tagged PDF using standard LaTeX packages will be available.

- Relevant document metadata can be specified and will be used to provide XMP-encoded data for the generated PDF, as required by certain PDF standards.

- Specifying alternate text for all relevant standard LaTeX commands and environments will be available.

- Attaching associated files to all relevant standard LaTeX commands and environments will be possible.

- Support for hyperlinking and outlines will be provided.

- LaTeX documents can be formatted to conform to certain PDF standards.

Reaching the above set of milestones does not mean that from this point onwards one can produce structured PDF from every LaTeX source; however, structured PDF output can be produced from any document source that loads only packages that have been updated to use the new standard interfaces (as produced by this project).

As the conversion of external packages (see task 2.4.3) is not part of the project scope as defined by this document, there will be further work necessary to broaden the set of LaTeX documents that can processed by a workflow for generating structured PDF output.

4   Resource assumptions and requirements

The project is assumed to be carried out by members of the LaTeX Project team, with 2 FTEs of their time being financed as part of the project's budget. There are, however, a number of (sub)tasks for which additional external help will be needed/helpful. The following is an (probably incomplete) list of such additional resource requirements.

Maintainers of external packages The assumption is that all the necessary updates to external packages are being done by their maintainers (if such maintainers exist) and will not be funded from the project resources. This assumption may not be valid in all cases.

Technical writers An important part of the project will be to provide high-quality documentation, for both users and package developers. While this documentation will be produced as part of the project tasks it would be helpful to get additional outside help through professional copy-editors/technical writers in order to improve the final results.

External experts For certain tasks, in particular the research tasks, participation of experts external to the LaTeX project team is assumed. Examples are Adobe DC engineers and Adobe researchers.

Testers The user and developer acceptance testing each requires a larger amount of external resources; see task 2.4.1 for a discussion of the required skills and needs.

Access to Adobe's knowledge base For many tasks the project team will need to consult documentation of PDF standards, Adobe products, APIs, etc. Some of these documents may be 'internal' or otherwise difficult to locate and access directly.

Software licences Although a lot can be done at minimal cost using Open Source software, other licences will probably also be needed for, at least, the following: Adobe software, Callas Software (pdfpilot), and other access tools and development tools.

References

Note that due to a bug in MacOS some links (those containing a #) may not work if clicked on from within this document. In that case copy them to your browser manually to access the external document.

[1] acro. Typeset acronyms. https://www.ctan.org/pkg/acro.

[2] Adobe Systems Incorporated. Document management – Portable document format – Part 1: PDF 1.7. 1st ed. July 1, 2008. https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf.

[3] array. Extending the array and tabular environments. https://www.ctan.org/pkg/array.

[4] arXiv.org. https://www.arxiv.org.

[5] David Carlisle, Patrick Ion, and Robert Miner. Mathematical Markup Language (MathML) Version 3.0 2nd Edition. Apr. 10, 2014. https://www.w3.org/TR/MathML3/.

[6] CTAN. The Comprehensive TeX Archive Network. https://www.ctan.org.

[7] LaTeX development formats are now available. Sept. 1, 2019. https://www.latex-project.org/news/2019/09/01/LaTeX-dev-format/.

[8] glossaries. Create glossaries and lists of acronyms. https://www.ctan.org/pkg/glossaries.

[9] hyperref. A package for extensive support for hypertext in LaTeX. https://www.ctan.org/pkg/hyperref.

[10] hyperxmp. Embed XMP metadata within a LaTeX document. https://www.ctan.org/pkg/hyperxmp.

[11] International Standard. ISO 14289-1:2014. Document management applications — Electronic document file format enhancement for accessibility — Part 1: Use of ISO 32000-1 (PDF/UA-1). 2nd ed. Dec. 2014. https://www.iso.org/obp/ui/#!iso:std:64599:en.

[12] International Standard. ISO 16684-1:2019. Graphic technology — Extensible metadata platform (XMP) specification — Part 1: Data model, serialization and core properties. 2nd ed. Apr. 2019. https://www.iso.org/obp/ui/#!iso:std:75163:en.

[13] International Standard. ISO 19005-2:2011. Document management – Electronic document file format for long-term preservation – Part 2: Use of ISO 32000-1 (PDF/A-2). 1st ed. July 2017. https://www.iso.org/obp/ui/#iso:std:iso:32000:-2:ed-1:v1:en.

[14] International Standard. ISO 19005-3:2012. Document management — Electronic document file format for long-term preservation — Part 3: Use of ISO 32000-1 with support for embedded files (PDF/A-3). 1st ed. Oct. 2012. https://www.iso.org/obp/ui/#iso:std:iso:19005:-3:ed-1:v1:en.

[15] International Standard. ISO 32000-2:2017(en). Document management — Portable document format — Part 2: PDF 2.0. 1st ed. July 2017. https://www.iso.org/obp/ui/#iso:std:iso:32000:-2:ed-1:v1:en.

[16] l3build. A testing and building system for (La)TeX. https://www.ctan.org/pkg/l3build.

[17] Leslie Lamport. LaTeX—A Document Preparation System—User's Guide and Reference Manual. Reading, MA, USA: Addison-Wesley, 1985. ISBN: 0-201-15790-X.

[18] latex. A document preparation system. https://www.ctan.org/pkg/latex.

[19] Frank Mittelbach et al. The LaTeX Companion. Second edition. Reading, MA, USA: Addison-Wesley, 2004, pp. xxvii + 1090. ISBN: 0-201-36299-6.

[20] pdfx. PDF/X and PDF/A support. https://www.ctan.org/pkg/pdfx.

[21] tagpdf. Tools for experimenting with tagging using pdfLaTeX and LuaLaTeX. https://www.ctan.org/pkg/tagpdf.

[22] The LaTeX Project. https://www.latex-project.org/.