The Mars Project — PDF in XML

Matthew R. B. Hardy
Adobe Systems Incorporated
345 Park Ave,
San Jose, CA 94086, USA
mahardy@adobe.com

ABSTRACT

The Portable Document Format (PDF) is a page-oriented, graphically rich document format based on PostScript semantics. It is the file format underlying the Adobe® Acrobat® viewers and is used throughout the publishing industry for final form documents and document interchange. Beyond document layout, PDF provides enhanced capabilities, which include logical structure, forms, 3D, movies and a number of other rich features.

Developers and system integrators face challenges manipulating PDF and its data. They are looking for solutions that allow them to more easily create and operate on documents, as well as to integrate with modern XML-based document processing workflows.

The Mars document format is based on the fundamental structures of PDF, but uses an XML syntax to represent the document. Mars uses XML to represent the underlying data structures of PDF, as well as incorporating additional industry standards such as SVG, PNG, JPG, JPG2000 and OpenType. Mars combines all of these components into a ZIP-based document container.

The use of open standards in Mars means that Mars documents can be used with a large range of off-the-shelf tools and that a larger population of developers will be very familiar with its underlying technology. Using these standards, publishers gain access to all of the richness of PDF, but can now tightly integrate Mars into their document workflows.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation — *Markup languages*; I.7.4 [Document and Text Processing]: Electronic Publishing.

General Terms

Documentation

Keywords

PDF, Mars, XML, SVG, Zip, Package.

1. INTRODUCTION

The Portable Document Format (PDF) [1] was introduced by Adobe Systems Incorporated in 1993. The primary goal of PDF was to perfectly reproduce the appearance of a document, capturing the author's intent completely. The page content of PDF is based on a static subset of PostScript® [2].

PDF has become the de facto document format in the publishing industry and many other areas. There are a number of reasons PDF has been so widely adopted. It is platform independent, and can be used for safe and secure document interchange. Content can be protected within a PDF document. It is a highly compact format, providing efficient access to any component of the document.

Adobe provides PDF as a royalty-free specification, allowing third parties to build applications. Specific subset of PDF, such as PDF/A [3] and PDF/X [4] are already ISO standards and the entire PDF specification has just been sent to ISO for standardization.

PDF contains many features that are beyond simple print reproduction. The list below highlights some of the enhanced features that PDF supports:

- Document Navigation (e.g. bookmarks/links)
- Fillable Forms and XFA
- Logical Structure
- Accessibility/Content Access
- Video and Sound
- Annotations/Collaboration
- Layers and 3D objects
- Signatures/Security
- Document Packages

PDF has clearly evolved to meet needs beyond its initial conception. However, there are a number of characteristics that are desirable beyond those currently available.

PDF would benefit from more third party tools and developers. It would also be beneficial for PDF to natively support more integration with open standards.

As we move to a world where publishing workflows are entirely XML-based [5], it would be highly desirable for PDF to fit into those workflows natively. The goal of the Mars projects was to do just that. Mars provides an alternative representation for PDF using XML and open standards.

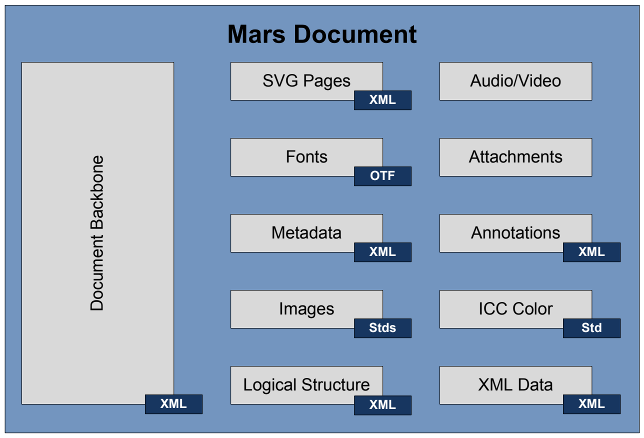

Figure 1 – Mars Concepts.

Figure 1 lists the primary components of a Mars document and the formats used to represent them. Mars uses custom XML Tagsets to represent PDF specific features, but uses industry standard formats whenever possible.

Page content is defined using SVG [6], a World Wide Web Consortium (W3C) XML-based vector graphics language. Images can be represented using the following images standards PNG, JPEG, JPEG2000 or JBIG2. Fonts are represented using the industry standard OpenType format. Color is managed using International Color Consortium (ICC) color profiles.

2. PDF DOCUMENT STRUCTURE

Before looking at how Mars is used to provide an alternative representation for PDF, it is important to understand how a PDF is structured.

PDF can conceptually be viewed at a number of different levels. There is the underlying syntax used to represent PDF on disk; there is the structured view, which looks at the objects and data structures of PDF; finally there is a high level view looking at how these objects are combined to represent the different aspects of a PDF document.

2.1. Syntax and Object Structure

At the lowest level, a PDF document consists of a set of objects, which can reference other objects. An example of the syntax used to represent these objects is:

```
1 0 obj << /Key (Value) >>
```

Each object has an object number which can later be used to reference the object. Objects also have what is known as a generation number, which is used to simplify updating the document, but is not of relevance here. In this example, the object number is 1 and the generation value is 0.

The most common object type in PDF is the dictionary object, which is shown in this example. A dictionary is an unordered set of key-value pairs. The '<<' and '>>' are used to represent a dictionary and the entries inside are pairs of objects consisting of a **name** object as the key (a **name** is represented using a preceding '/' character followed by a string) and any type of object as the value. In the example above, the key is '`/Key`' and the value object is a **string** '`Value`'.

PDF allows objects to reference other objects, to allow for more complex data structures. For example, if another object wanted to use the object defined previously, it would simply include "`1 0 R`". An example of this might be:

```
2 0 obj << /InlineDict 1 0 R >>
```

The above dictionary has a key of '`/InlineDict`' whose value is the object defined previously.

The fundamental types that exist in the PDF object structure are dictionaries, arrays, strings, integer and real numbers, Boolean values and streams. Streams are similar to dictionaries, but are used to hold binary data or content streams (which represent page content). They consist of a dictionary component and raw data.

At a slightly higher level, a PDF document can be viewed as a hierarchy of dictionaries. These dictionaries are shared in multiple places within the document using the referencing mechanism described above.

PDF can therefore be thought of as a graph of objects which is comprised of multiply linked trees with shared nodes. While the page content is stored in the streams, the meta-information that represents a PDF and allows for the richness of PDF is stored in these trees.

2.2. PDF Document Structure

But how is the object/dictionary structure described above used to define a document? The diagram in Figure 2 shows the primary component dictionaries that are used to form the basis for a simple PDF document.

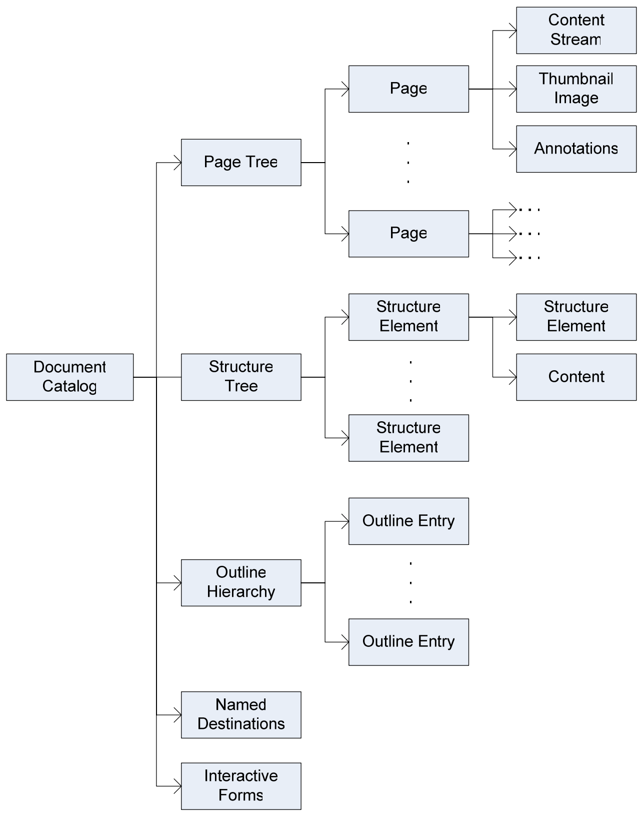

Figure 2 – PDF Document Outline.

At the top level of a PDF document, there is the Document Catalog Dictionary.

2.2.1. Document Catalog Dictionary
The document catalog provides the primary entry point into the document and is the container of all the sub-trees that make up a PDF document.

It also contains meta-information about document, which includes the following: the version of the PDF (latest version being 1.7 [7]); the document layout (e.g. single column or multi-column); document metadata; language identifiers; and more…

2.2.2. Page Tree
The page tree contains dictionaries representing each page. Each Page dictionary provides information about the page. This includes include the size of the page, its rotation, any other metadata associated with the page, etc..

Most importantly, it contains an entry containing the (content) stream representing the page contents and a resource dictionary, which in turn contains any resources for the page (e.g. images, fonts, etc.).

2.2.3. Content Streams
As described previously, streams can be used to contain any type of raw data, but a content stream specifically refers to a stream holding page content.

This page content uses a PostScript-like notation to draw graphical content. The PDF content operators are not identical to the PostScript operators, but utilize the same reverse polish notation and provide similar levels of functionality. However, unlike PostScript, there is no support for conditional statements or loops. The PDF content operators can display text, draw lines and curves, paint images, etc..

The resource dictionary holds all the resources that the page uses. Such resources include fonts, images, graphics states, shadings and patterns, etc..

2.2.4. Logical Structure Tree
The structure tree allows a PDF document to contain logical structure information about its contents [8]. The structure tree can be considered similar in nature to an XML document processed into a DOM [9], but with slightly restricted contents (i.e. no support for XML Processing Instructions or comments).

The structure tree consists primarily of structure element nodes. These are the equivalent of elements in an XML document. Just like in XML, element nodes can have attributes. The children of a given element node can contain more elements, but can also refer to page content.

A content stream can contain marking operators, which can be used to give that content an ID (which is a numerical value unique to that content stream). An element node in the structure tree indicates which content stream it refers to (e.g. a specific page) and the ID associated with the content. This links the structure to the page contents[1].

2.2.5. Bookmarks/Actions
The outline hierarchy dictionary in the catalog is used to create bookmarks for the document. Again, it contains all the bookmarks for the document. The hierarchy is used to determine containment, so when creating bookmarks for a long document, each chapter might have a top level entry and then subsequent child entries.

The bookmarks specify an action that will be performed when they are selected. An action can perform a number of different types of task. The most common action linked to a bookmark is a '`GoTo`' action, which is used to move to a specific location within the document.

However, more complex actions do exist and are used in a number of places other than for bookmarks. Actions can be used to play sounds or movies, launch external programs, execute JavaScript, reset forms, etc..

2.2.6. Named Destinations
Related to bookmarks and actions are named destinations. Rather than have a destination in a '`GoTo`' action point to an area on a page, a named destination can be defined.

A name tree is used to store this data and provide fast access to the destinations. A unique name is linked to a dictionary containing the destination information.

It is useful to have these definitions when a destination is going to be used in multiple places. Although the bookmarks are the primary user of the name tree, there are other uses (e.g. links on a page can make use of named destinations).

2.3. Overview
The above descriptions provide a view of PDF that is very much a set of structured trees, which are used to describe a PDF document. In fact, the dictionaries and trees map very well to the XML model, because they are intrinsically hierarchical in nature.

The main difficulty with integrating PDF into modern XML workflows is with the underlying PDF syntax, not the conceptual model of a PDF. If PDF documents could be described in XML, rather than the current PDF syntax, this would facilitate the integration of PDF into such workflows.

Of course, doing this is not trivial. While large sections of PDF do map well to XML, other sections don't necessarily map as well in their current form (e.g. logical structure). PDF also has a number of optimization that work well for the PDF syntax (or were necessary when the format was updated to add new features), but would be better if done differently in XML.

3. MARS
The Mars format is a packaged XML representation of PDF. The Mars document structure generally mirrors the PDF document structure where possible, but uses XML to represent the dictionary structures and page streams of PDF. Where this was not possible, these changes are explicitly stated in the following sections.

In cases where PDF uses binary data as a component of the document (e.g. an image or font), Mars uses industry recognized standard file formats to represent them.

Because of the need for multiple XML tagsets to represent the different components of the document, coupled with the

[1] For a more in-depth description of logical structure in PDF see (10).

requirement to store binary data within the document, Mars uses multiple sub-files, stored in a packaged zip [10] format to contain all these disparate components.

From a high level perspective, a Mars document is comprised of multiple file-based components (XML and binary), which interlink to produce a connected set of sub-files, which in turn represent the overall document. URIs are used to connect the sub-files within a Mars document. The linking connectivity used within Mars is generally unidirectional.

Although the XML specification provides a mechanism for including binary data inline in an XML document, the mechanism is inelegant and increases the complexity and size of the document. It also makes it harder to extract parts of the document. For these reasons, any non-private binary data within Mars is stored in separate sub-files.

3.1. Packaging

In section 3.2 the reasons for splitting the dictionary structures into multiple XML files are discussed. However, it is first necessary to discuss how a Mars document can contain multiple sub-files. As described previously, Mars uses a zip-based package to contain these sub-files. The specific implementation uses the Universal Container Format [11].

Mars uses a standard zip format to package the files. However, the packaging used by Mars imposes a number of requirements on the contents of the zip. The primary requirements are as follows: the first file in the package must be the '*mimetype*' file, which must be at the root of the package (i.e. not in a subfolder), that contains the mime type for the package contents; a '*container.xml*' in the '*META-INF*' folder describing any associations in the package (optional if none exist); and the file 'backbone.xml' which is the entry point into the Mars document and is not optional.

By using a zip package, Mars gains a number of advantages. The first of these is that many of the Mars components, especially the XML components, compress well with zip. The second advantage is that zip provides random access to the components within the Mars package. The nature of the zip package also facilitates deferred loading of document components (e.g. access over the Internet where the document can be partially displayed before the entire file has been downloaded).

Other than the requirements above, Mars does not explicitly define a strict package layout. It is up to the creating application to decide on file locations. However, the Mars specification provides a recommended default layout, which creating applications can choose to follow.

3.2. Mars Components

Mars splits the dictionary structure of PDF into multiple XML sub-files. These files represent different aspects of the overall document. Figure 3 shows an overview of the sub-files that can be found inside a Mars package, showing the common components found within a Mars document package. The following sections describe these components.

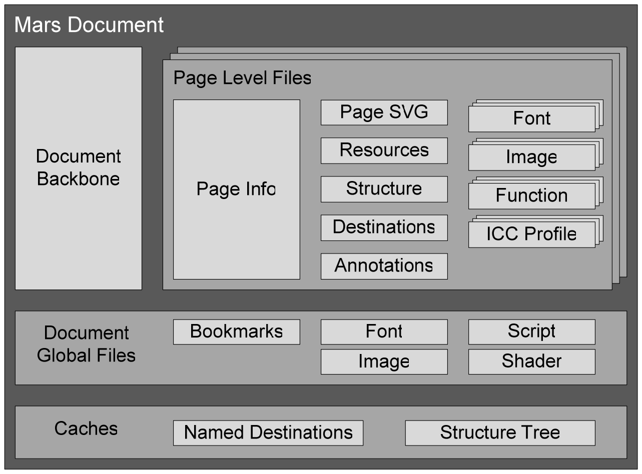

Figure 3 – Mars Document Outline.

3.2.1. Backbone

At the top level of a Mars package must be the *backbone.xml* file. This file provides a very similar function conceptually to the catalog dictionary in a traditional PDF. It is the root of the document. It specifies the primary entry points into the rest of the document components and provides any requisite information about the document.

The backbone provides information as to whether a given tree from the PDF representation is present and provides a URI to its source (note that the URI is relative to the package). These entry points include links to each of the pages within the document, the bookmarks, metadata, article threads, etc..

A difference between the catalog and backbone is in the page references. In the catalog, a reference is made to the top of the pages tree and the tree must be iterated to find a specific page. In the backbone, this information is provided explicitly and the bounding box of each page is given at this point. This allows a consuming application to list the number of pages and know their size before reading any sub-files.

The backbone contains other information as well. It contains language information, version information, whether or not the document is logically structured and some other meta-information about the document.

3.2.2. Pages Level Files

Each page is represented at the top level by an *info.xml* file. These page info files can be considered to be equivalent to the page dictionaries in PDF.

While the backbone holds the bounding box size for each page, the page info holds the crop box, bleed box, art box, etc.. It also has other information, such as the page rotation, any preferred zoom, and others…

However, it also provides a similar purpose to the backbone, in that it is the entry point into the content that combines to form a page. It holds URIs for the SVG representing the page; to the content annotations; to the logical structure for the page; and for the destinations on the page. Unlike the page dictionary in PDF, it does not directly hold a link to the resources for the page, because these are directly referenced by the SVG page contents.

3.2.2.1. *Page SVG*

The page info file refers directly to an SVG file describing the page. This SVG file is very similar in concept to the page content stream in PDF, but uses the W3C scalable vector graphics language to represent the page contents, rather than PDF operators.

Mars uses a subset of SVG Tiny [6] as its core. Because SVG has some aspects which are less relevant to a page description format, these features are not supported in Mars. The table below shows the primary SVG features excluded from Mars and the reason they were excluded (see the Mars reference for more information [11]).

Table 1 – Major SVG Tiny Features Excluded from Mars

Name	Description/Reason
CSS	All CSS-related markup is not supported. SVG provides other equivalent mechanisms, so CSS was excluded for simplicity.
Interactivity and Scripting	The SVG used in Mars is intended to be static. Mars provides the PDF model for scripting.
Animation and Multimedia	PDF already provides mechanisms for rich media and page content is designed to be static.
textPath and tref	These were not supported because PDF uses an alternative approach.
Color Interpolation	Conflict in color models between SVG and PDF.

Similarly, there are a number of PDF features that are current beyond the capabilities of SVG. For these Mars defines extensions to SVG to support these enhanced capabilities. These extensions consist of XML elements and attributes interleaved with the SVG in the Mars namespace.

There are too many extensions to describe here, but features that are added include transparency extensions, soft-mask enhancements, high-end print support, image extensions, smooth shading, blending and font descriptors.

The other primary addition is marked content groups. PDF allows content to be 'marked' for use by a consuming application or plug-in. The marked content mechanism allows content to be marked for later identification and provides a mechanism for associating metadata with that content. There are many uses for this and it is common for consuming applications to add private marked content to the page. A formal use of marked content is for logical structure, which is described later.

Note, one major difference between the PostScript/PDF graphics model and the SVG graphics model is that they use different coordinate spaces. PDF has its origin in the lower left corner and the *y* coordinate grows upwards. SVG has its origin in the upper left corner and grows the *y* coordinate downwards. When converting between Mars and PDF, this has to be taken into account.

3.2.2.2. *Page Resources*

Although SVG provides a rich vector graphics format, there is still a need for extra data. This includes fonts, image, shadings and patterns. PDF uses the page dictionary to hold both the content stream and its resources in separate dictionaries and then provides mechanisms to let the page content access these resources by name. Mars instead simply allows the SVG page content to directly refer to the resources it uses.

As described previously, Mars does not enforce a specific package layout beyond some basic requirements. Therefore, an application creating a Mars document can choose where to place the resources inside the package. However, the recommendation is to put shared resources in a shared folder, but to put resources that are only used once in the same location as the SVG referencing them.

Fonts are stored in OpenType and images can use a number of standard formats including JPEG, JPEG2000, PNG and JBIG2.

3.2.2.3. *Annotations*

Each page can have annotations associated with it. There are two classes of annotation, content and markup. Markup annotations might be things such as text, circles, and underlines and are generally used in markup/review of a document. The content annotations are used to actually add or enrich the content of a document. These include such items as movies, links, sounds, 3D and watermarks. However, form fields are perhaps the most important use for content annotations.

Because content annotations are considered part of the page, they are directly referenced from the page info file. All the content annotations for a given page are within a single XML file.

Markup annotations are different, in that they can be considered as existing outside of the page. While they often appear on the page, they are not usually an integral component of the page. It is quite common to turn them off or hide them. Because of their nature, markup annotations are not explicitly linked from the page. Instead, an implicit association is created between a given page and it markup annotations by the presence of '*pg.svg.ann*' file in the same location as the '*pg.svg*' file describing the page.

This implicit style of association makes it very easy to add and remove markup annotations without changing any other component of the document, including the page. This is of particular benefit when annotations are used for reviewing purposes, where they do not in fact change the underlying document.

Annotation can have very complex appearances. PDF allows content streams to be used to describe the appearance of annotations. Mars provides a similar mechanism, but just as for page content, annotation appearances are defined using SVG and use the same SVG features supported for pages.

3.2.2.4. *Destinations*

Unlike PDF where destinations are stored in a single name tree at the top level of the document, Mars tries to make the document more modular. If a new page is added, it is important to have to make as few updates to the rest of the document as possible. With this in mind, the decision was made to migrate the destinations from their centralized location to the page they belonged to.

When a new page is added, if there are any destinations associated with that page, the page info file must reference them. However, no update is required to the rest of the document (see the section 3.2.4 for a description of how fast access to this information is provided within Mars).

3.2.2.5. *Logical Structure*

For similar reasons to those described above, logical structure is also distributed to the pages the referenced content belongs to. In PDF, the structure tree describes the logical structure and hierarchy of the document and uses references into the page content to link the structure to the content. Reading all the structure information at once can be problematic. Mars therefore places the structure in the '*struct.xml*' file at the page level.

However, unlike the destinations tree, whose hierarchy is solely used to provide fast search capabilities into the tree, the logical structure tree hierarchy must be preserved. Because there is no central repository for the structure information and nodes in the tree can have content spanning multiple pages, each structure file must contain information regarding all nodes above content belonging to the relevant page.

Because nodes can contain content on more than one page, they can occur in multiple files. Therefore, unlike a single tree where ordering can be determined simply by position of sub-nodes in the tree, a more complex mechanism is needed to determine the ordering. Mars uses a path mechanism, similar in concept to XPath, to provide the hierarchy and positioning information of nodes.

A path contains a sequence of stages, separated by a '/' character. Each stage is comprised of an optional prefix (which is a string), a numerical value and an optional postfix (which is also a string). If a prefix is present, it is separated from the numerical value by a colon. If a postfix is present, it is separated using a hyphen. An example is "abc:1.3-def". Ordering is determined by sorting the nodes by their path. Sorting is done by prefix first (if present), numerical value second (which must be present) and postfix last (again if present). This allows easy addition of nodes and facilitates merging of documents, where two nodes sets can be given prefixes, but their internal sort order remains the same. It is not valid to have an absolute zero value as the numerical component (e.g. 0, 0.0 or 0.0.0 are not valid, because this complicates the addition of later nodes should the document be changed).

In a very similar manner to PDF, content in the SVG is given an ID, which is then referenced from within the structure file. The structure uses the marked content group mechanism discussed previously to set this ID and reference it. This adds an SVG group element with an XML ID into the page contents surrounding the content that is to be marked. Because this is the only required change, the SVG needs to embed very little information about the logical structure.

3.2.2.6. ***Page Summary***
The Mars format attempts to keep all the components relevant to a specific page in one location. These components need very little information outside of the page files. This design principle makes each page relatively independent and therefore makes page addition/removal and general document construction simpler.

It also means that when reading in a Mars file for viewing, the viewer need only open the files relevant to the specific page and therefore minimize the amount of data it must read for each page or task.

3.2.3. Bookmarks
Because the bookmark tree is intended to provide a navigation mechanism for the entire document, it is important to be able to provide that information to a consuming application in an efficient manner. Therefore, Mars keeps the entire bookmarks tree in one XML file.

3.2.4. Caches
For both page named destinations and logical structure, Mars moves the data to exist alongside the page it belongs to, rather than holding it in a single file.

One reason for moving the data to the page level was to simplify document construction. However, for both destination and structure, another reason to separate these out was due to the large amount of data related to them. The destinations tree can be very large, with large documents having many thousands of named destinations (e.g. the current PDF reference manual has over 70,000 destinations). By moving the entries to a per page basis, only destinations meaningful to the components of the document that are in use need to be processed.

However, it is sometimes necessary to access every single entry, but only for a small amount of the information actually present in each entry. This is entirely possible in Mars, but it is highly inefficient. Therefore, the caches were introduced to Mars. These provide access to some of the basic information in the document without having to search the entire Mars file. However, they can be entirely regenerated or replaced if they are not present or become out of date from the data inside the Mars file.

3.2.4.1. ***Structure Cache***
The main purpose of the structure cache is to be able to reproduce the outline of the entire structure tree without reading the per-page structure files.

It does not provide any information beyond the hierarchy of the nodes. It contains no names for nodes or such. It does provide URIs back to the per-page structure files so that after it has been read in, the nodes easily be linked to their full information.

Again, to give an idea of the size of the information present, a large document such as the PDF reference manual has over 200,000 element nodes and over 125,000 content references.

While the cache can contain element nodes, because of the nature of the structure path mechanism described previously, element nodes are optional and only content nodes need to be present. The reasoning behind this is that it is possible to regenerate all the cache information from just the leaf nodes, so by keeping the size of the cache as low as possible, we increase performance and scalability.

3.2.4.2. ***Named Destinations***
The named destinations cache simply stores the name of each destination and the page it is on. This optimizes searching for named destinations within a Mars document.

3.3. Mars Conclusion
The packaged nature of Mars has been described in the previous sections and each of the components which make up a Mars file have been discussed.

However, very little of the actual tagset used by Mars has been described and how it compares to PDF. The following section describes how we actually construct a Mars file.

4. HOW TO BUILD A MARS DOCUMENT
This section takes a simple PDF document and shows how it looks as a Mars document. It then describes how we would add bookmarks and logical structure to that simple document including adding the extra information to the SVG.

4.1. Simple PDF in Mars Format

The simple PDF consists of a single page, with the text "Hello World" and "Goodbye Universe!" displayed. Figure 4 shows the PDF syntax to represent such a page.

```
%PDF-1.2
1 0 obj
<<
/Type /Catalog
/Pages 2 0 R
>>
endobj

2 0 obj
<<
/Type /Pages
/Kids [3 0 R]
/Count 1
>>
endobj

3 0 obj
<<
/Type /Page
/Parent 2 0 R
/Resources 4 0 R
/Contents 5 0 R
/MediaBox [0 0 612 792]
>>
endobj

4 0 obj
<<
/ProcSet [/PDF /Text]
/Font <</F1 6 0 R >>
>>
endobj
```

```
5 0 obj
<<
/Length 51
>>
stream
  BT
  /F1 24 Tf
  1 0 0 1 260 600 Tm
  (Hello World) Tj
  1 0 0 1 260 550 Tm
  (Goodbye Universe!) Tj
  ET
endstream
endobj

6 0 obj
<<
/Type /Font
/Subtype /Type1
/Name /F1
/BaseFont /Helvetica
>>
endobj

trailer
<<
/Root 1 0 R
>>
```

Figure 4 – PDF "Hello World" Example.

As described previously, at the root is the catalog dictionary (`1 0 obj` in this example). It refers to the pages tree and since this document has no other enhanced features, this is all it contains. The pages tree (`2 0 obj`) keeps a count of the number of pages and has an array of pages (in this case, an array with one entry). The page (`3 0 obj`) references its resources (`4 0 obj`) and contents (`5 0 obj`) and specifies its `MediaBox`. The page itself consists of a stream displaying the text "Hello World" at position (*260,600*) and "Goodbye Universe!" at (*260,550*) using the font `/F1`. This font is defined in the resources and actually described in the font dictionary (`6 0 obj`).

Table 2 – Files in Simple Mars

Path	File Name
/	mimtetype
/	backbone.xml
/page/0/	info.xml
/page/0/	pg.svg

If we create an equivalent Mars file for this, we would expect to see the set of files listed in Table 2 within the Mars package. The contents of the Mars files (excluding '*mimetype*' which is pre-defined) would be as seen in Figure 5.

- /backbone.xml

```
<PDF PDFVersion="1.4" Version="0.8.0">
 <Pages>
  <Page src="/page/0/info.xml"
        x1="0" y1="0" x2="612" y2="792" />
 </Pages>
</PDF>
```

- /page/0/info.xml

```
<Page>
 <Contents src="pg.svg"/>
</Page>
```

- /page/0/pg.svg

```
<svg fill="none" stroke="none">
 <defs>
  <font-face font-family="F1">
   <font-face-src>
    <font-face-name name="Helvetica"/>
   </font-face-src>
  </font-face>
 </defs>
 <text font-size="24" font-family="F1"
       fill="rgb(0,0,0) device-color(DeviceGray,0)"
       fill-rule="evenodd">
  <tspan x="260" y="192">Hello World</tspan>
  <tspan x="260" y="242">Goodbye Universe!</tspan>
 </text>
</svg>
```

Figure 5 – Mars "Hello World" Example.

The backbone contains a reference to the first page and defines its bounding box. Other than that it notes that we had a PDF version 1.4 compatible document and a Mars version 0.8.0 document.

The page info file is similarly simple. Note that because the pg.svg file is at the same level in the package as the page info file, we simply use a relative reference to it, not having to explicitly state the absolute path. Should this page ever be moved to another document or moved within the package, only the backbone would need to be informed of the change.

Finally, we have the SVG file representing the page contents. Note that because the font wasn't actually embed in the PDF, but instead used a named system font (one of the default fonts), Mars also doesn't need to embed it. Therefore, the font definition is entirely inline with the SVG. The SVG consists of a `<defs/>` section, which names the font (if we had more than one line of text, we would want to re-use this resource, hence its definition here, rather than inline with the text).

Finally we actually display the text. As described in section 3.2.2.1, the coordinate spaces of SVG and PDF/PostScript differ. The example shows how the coordinates for "Hello World" in the PDF are translated to SVG coordinates. Given that we have defined the page to be 612 units wide by 792 units high, the PDF coordinates for "Hello Word" of (*260,600*) translate to (*260,192*).

4.2. Enhancing the Document

The above example is very simple and doesn't show how modular Mars is. However, there isn't the scope in this paper to describe every feature of Mars and how to create a Mars document using it. Therefore, the example below is limited to adding bookmarks and logical structure to our existing Mars document.

4.2.1. Bookmarks

To add bookmarks using named destinations to the sample Mars file, we would need to add or update the files seen in Table 3.

Table 3 – Adding Bookmarks

Path	File Name
/	backbone.xml
/	bookmarks.xml
/page/0/	dests.xml

The contents of these files would be as seen in Figure 6 (note that new or changed content is highlighted).

- /backbone.xml

```
<PDF PDFVersion="1.4" Version="0.8.0">
 <Bookmarks src="bookmarks.xml"/>
 <Pages>
  <Page src="/page/0/info.xml" ID="0"
        x1="0" y1="0" x2="612" y2="792" />
 </Pages>
</PDF>
```

- /bookmarks.xml

```
<Bookmarks Open="true">
 <Bookmark>
  <Title>Hello World</Title>
  <Action>
   <GoTo><Dest Name="Hello"/></GoTo>
  </Action>
 </Bookmark>
</BookMarks>
```

- /page/0/dests.xml

```
<Destinations>
 <Dest Name="Hello">
  <XYZ Left="246" Top="65" Zoom="4"
       Page_ref="/backbone.xml#0"/>
 </Dest>
</Destinations>
```

Figure 6 – Mars Bookmarks.

The *bookmarks.xml* file specifies the name for the bookmark and the action to be performed when the bookmark is selected. In this case, it is a `GoTo` action. The place to go to is referenced using a name 'Hello'. Although only one bookmark is defined in this file, it is simple to add more. Under the `Bookmarks` element, it is legal to have any number of `Bookmark` elements. Each `Bookmark` element can contain `Bookmark` elements as well, allowing for the nesting/hierarchy of bookmarks.

The *dests.xml* file at the level of the page contains the actual destination information. Note that the *bookmarks.xml* file does not explicitly have a reference to the *dests.xml* file. The place to look up destinations names is implicit.

Finally, we declare the bookmarks by adding a reference to them in the *backbone.xml* file. There is one other change necessary in the backbone however. For the destination to be able to reference the page, it must refer to the node in the backbone that is relevant to it. For this it uses a fragment identifier referencing an ID. Therefore, an ID was needed on the page node being referenced.

Apart from changes to the backbone to include the bookmarks and add an ID (which in most cases would be present by default because page IDs may be used elsewhere), the rest of the package is left untouched, with only new files having been added.

4.2.2. Logical Structure

The addition of logical structure requires a few more changes to existing files than the addition of bookmarks. This is because the logical structure mechanism needs to refer to content on the page.

The simple logical structure tree we want to add is shown below in Figure 7. The top level of the tree is a 'Sect' node (section), with two children, both 'P' nodes (paragraphs), which point into the content.

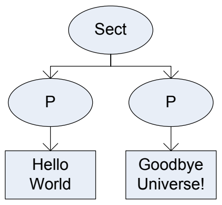

Figure 7 – Logical Structure.

The files that would need to be added or changed to add this logical structure are listed in Table 4.

Table 4 – Adding Logical Structure

Path	File Name
/	backbone.xml
/page/0/	info.xml
/page/0/	pg.svg
/page/0/	struct.xml

The contents of these files are described below in Figure 8 (again highlighted to show the changes).

- /backbone.xml

```
<PDF PDFVersion="1.4" Version="0.8.0">
<Marked TaggedPDF="true"/>
<Bookmarks src="bookmarks.xml"/>
 <Pages>
  <Page src="/page/0/info.xml" ID="0"
        x1="0" y1="0" x2="612" y2="792" />
 </Pages>
</PDF>
```

- /page/0/pg.svg

```
<svg fill="none" stroke="none">
 <defs>
  <font-face font-family="F1">
   <font-face-src>
    <font-face-name name="Helvetica"/>
   </font-face-src>
  </font-face>
 </defs>
 <g pdf:Mark="P" ID="Para1">
  <text font-size="24" font-family="F1"
        fill="rgb(0,0,0) device-color(DeviceGray,0)"
        fill-rule="evenodd">
   <tspan x="260" y="192">Hello World</tspan>
  </text>
 </g>
 <g pdf:Mark="P" ID="Para2">
  <text font-size="24" font-family="F1"
        fill="rgb(0,0,0) device-color(DeviceGray,0)"
        fill-rule="evenodd">
   <tspan x="260" y="242">Goodbye Universe!</tspan>
  </text>
 </g>
</svg>
```

- /page/0/info.xml

```
<Page>
 <Structure src="struct.xml"/>
 <Contents src="pg.svg"/>
</Page>
```

```
•   /page/0/struct.xml
<Structure>
 <Elem Lang="EN-GB" Path="/1" Tag="Sect" />
 <Elem Page_src="info.xml" Path="/1/1" Tag="P" />
 <MCR Path="/1/1/1" ref="pg.svg#Para1" />
 <Elem Page_src="info.xml" Path="/1/2" Tag="P" />
 <MCR Path="/1/2/1" ref="pg.svg#Para2" />
</Structure>
```

Figure 8 – Mars Logical Structure.

The structure tree defined in the *struct.xml* file creates the three elements that constitute the structure tree and two marked content references, which link these nodes to the page content.

A structure element in the structure tree that directly references page content must point to the containing page (i.e. "Page_src='info.xml'"). Containment is not defined by element hierarchy, but entirely through the use of the 'Path' attribute. Note that MCR nodes also have a path associated with them. This is because Elem nodes can contain multiple MCR nodes, so the content ordering has to be explicit also.

By using the path mechanism, rather than storing the structure tree, we instead store a description of the structure tree. The description is more amenable to incremental processing and modification.

To support these changes, the *pg.svg* file has been updated to provide information about the content using the marked content group mechanism. An ID has been added to the content and a mark associated with it. This is then referenced from the *struct.xml* file.

The *info.xml* file for the page has been updated to indicate that the page contains structure and the location of the *struct.xml* file. The backbone has similarly been updated to indicate that a viewer should look for structure.

4.2.3. Caches

Both of the updates we made can benefit from caches. Table 5 below lists the names and locations of the cache files that would be generated for the destinations and logical structure added previously.

Table 5 – Cache Files

Path	File Name
/cache/	names.xml
/cache/	struct.xml

The contents of these files are show in Figure 9.

The caches specify a query, which allows them to select every file containing structure should the data become stale.

In the case of the named destination cache, each destination name is listed. However, rather than repeat its location for each entry, this is moved to a containing `Dests` element, which groups all the destinations contained in that file.

A similar approach is taken with logical structure. The information that is shared between each component is moved outside of the individual entries and shared between all of them.

```
•   /cache/names.xml
<Cache>
 <Query>
  <Files><Pattern Value="/page/*/dests.xml"/></Files>
 </Query>
 <Data>
  <Dests Page="/page/0/dests.xml">
   <Dest Name="Hello"/>
  </Dests>
 </Data>
</Cache>

•   /cache/struct.xml
<Cache>
 <Query>
  <Files><Pattern Value="/page/*/struct.xml"/></Files>
 </Query>
 <Data>
  <Group ref="/page/0/struct.xml">
   <MCR Path="/1/1/1" />
   <MCR Path="/1/2/1" />
  </Group>
 </Data>
</Cache>
```

Figure 9 – Mars Caches.

For both destinations and logical structure, only partial information is placed within the cache. This information provides the data necessary to a consuming application. In the case of the destinations, it is just the name of the destination and the location of the rest of the data. This provides faster searching for named destinations within a Mars document. For logical structure, just the MCR leaf nodes are cached. This is because the paths from these are enough to reconstruct the structure tree outline and again to then reference the full per-page information when that becomes needed.

4.3. Creating Mars Files Conclusion

This section has described the process of creating a basic Mars file and then adding a number of enhanced features to it. The process is relatively modular, but a small number of updates are required to existing components to add these features.

By being modular, we minimize these updates and simplify the process of adding new content to an existing Mars document. There are other components (markup annotations) where no updates are required, because the files are entirely self-contained.

5. CONCLUSION

The previous sections have discussed Mars, an XML-based file format for representing PDF documents. A comparison was made between PDF and Mars and then the process of creating a Mars file to represent a given PDF was described. The goal was to highlight similarities and differences between PDF and Mars.

5.1. Goals for Mars

One goal for Mars was equivalence with the PDF file format. Every aspect of PDF must be representable using Mars. However, while the feature set is equivalent, the approach to representing each component can differ.

Mars makes document assembly modular and minimizes the amount of interconnectivity that is required. This simplifies document composition and the addition/removal of content. Mars also makes it relatively simple to add extra meta-information (e.g. logical structure) to an existing document. For example, adding page navigation through the use of bookmarks simply requires the addition of a bookmarks file, a destinations file at the page level (if it uses named destinations) and a single element added to the backbone. However, like PDF, the goal of mars is to be a final-form page description language. While it is not intended to directly benefit such uses as content reflow and iterative updates to the same document, the use of industry standard file formats to represent the underlying page structure does indirectly simply this process. The modularity in Mars mainly benefits the addition of semantic components (e.g. the addition of logical structure, bookmarks, article threads, etc.).

This modular approach has a number of other advantages over a less modular one. One of these is scalability. As a Mars file grows in number of pages, the amount of information that needs processing to load a single page does not increase (e.g. by keeping logical structure information in a separate file from the page information, this only needs to be processed when the logical structure needs to be consumed by a viewing application). Similarly, should we desire to add markup annotations to a page, we need only add a single annotations file to the page.

There are a number of other goals for Mars, including a desire for more tools to manipulate such documents. By using XML to represent Mars, we can leverage the huge number of XML parsers, validators, transformation and querying languages (e.g. XSLT/XQuery) that already exist.

Mars also makes it simpler to integrate the capabilities of PDF into existing XML workflows. Again, using standard tools, the data in Mars can easily be created or accessed a re-used inside these workflows. Mars also lowers the learning curve for developers who are not familiar with the underlying PDF syntax, but who are very familiar with XML and web standards.

The use of SVG to represent page content means that existing SVG tools can be used to create or process the pages in a Mars document. SVG, being an application of XML, already defines a DOM representation, which makes it easy to manipulate with existing tools. A number of other technologies have previously been created to convert PDF documents into SVG. However, the intent of many of these tools is to capture only the graphical content of a PDF and translate it into SVG. They do not capture the semantics or other information contained within the document (e.g. bookmarks, logical structure, layers, etc.). There are exceptions to this, such as PDF2SVG [12], which capture more of the semantic information. However, PDF2SVG still online captures a small subset of the semantic information present in PDF and is generally limited to reproducing appearance.

Tying all the benefits described above together, we have the zip packaging. This decreases the file size, while providing random access to the Mars components. The modular nature of Mars combined with the random access make Mars a very Internet-friendly format, where content can be loaded on the fly without downloading the entire package.

5.2. Future

At present an Adobe Acrobat plug-in can be obtained [11] which allows Acrobat to generate Mars files and to view them. This plug-in is still in development, but the goal is to have Acrobat interact with the Mars file format as seamlessly as it does with PDF.

Automatic generation of Mars from XML workflows allows easier access to the power of PDF, without losing any of PDF's capabilities. All combined, Mars is a file format with the same capabilities as PDF, but with a greater ability to integrate with the today's modern world.

6. ACKNOWLEDGEMENTS

I would like to thank Phil Levy, Principal Scientist and manager of the Mars Project at Adobe for supporting me in my day to day work and with writing this paper.

7. REFERENCES

[1] Adobe Systems Incorporated, *PDF Reference*, 5th ed., Berkley: Peachpit Press, 2005. 0-321-30474-8.

[2] Adobe Systems Incorporated, *PostScript Language Reference Manual*, 3rd ed., Menlo Park: Addison-Wesley, 1999. 0-201-37922-9.

[3] International Organization for Standardization, *Document management — Electronic document file format for long-term preservation — Part 1: Use of PDF 1.4 (PDF/A-1)*, 2005. ISO 19005-1:2005.

[4] International Organization for Standardization, *Graphic technology — Prepress digital data exchange — Use of PDF — Part 1: Complete exchange using CMYK data (PDF/X-1 and PDF/X-1a)*, 2001. ISO 15930-1:2001.

[5] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0*, 4th ed., 2006. [Online] http://www.w3.org/TR/2006/REC-xml-20060816/

[6] World Wide Web Consortium, *Mobile SVG Profiles: SVG Tiny and SVG Basic*, 2003. [Online] http://www.w3.org/TR/SVGMobile/

[7] Adobe Systems Incorporated, *PDF Reference*, 6th ed., San Jose: Adobe Systems Incorporated, 2006.

[8] M. R. B. Hardy, D. F. Brailsford, "Mapping and Displaying Structural Transformations between XML and PDF", in *ACM Symposium on Document Engineering*, 2002, pp. 95-102.

[9] World Wide Web Consortium, *Document Object Model (DOM) Level 3 Core Specification*, 2004. [Online] http://www.w3.org/TR/DOM-Level-3-Core

[10] PKWare Incorporated, *Zip Format Specification Application Note*, 1989. [Online] http://www.pkware.com/index.php?option=com_content&task=view&id=64&Itemid=107

[11] Adobe Systems Incorporated, Mars Project, 2006. [Online] http://labs.adobe.com/technologies/mars

[12] PDFTron Systems Incorporated, PDF2SVG, 2007. [Online] http://www.pdftron.com/pdf2svg/