PDF Creation Date:

May 31, 2007

Preliminary Mars File Format Specification

Version 0.8.0 4/23/2007

ADOBE DRAFT SPEC

Table Of Contents

1 Abstract

Mars is a representation of PDF documents that leverages standard representations for document information by combining XML, image, fonts, and color standard formats with Zip packaging to combine all document information into a single file.

The information in Mars is the same information that appears in PDF and organized in a similar, but not identical way. The Mars format makes component subassemblies of the PDF more independent and easier to manipulate as XML.

Mars is intended to be similar in size and performance to current PDF.

2 Introduction

PDF today faces increased demand to interoperate with XML based formats and technologies. PDF has historically been difficult for developers to work with because of its complex internal structure.

Mars addresses these issues. We hope to enable a larger group of developers to more easily build PDF-based applications. Mars does this by providing a representation of PDF that can be more easily understood and manipulated by XML-savvy developers and tools. Documents and forms can be created and manipulated in Mars format which can be directly opened by Acrobat(R) or Reader(R) or converted to PDFC (PDF Cos-based) format. (COS is the name of the object syntax used by PDF.)

The goals of Mars are to

1. Provide an XML representation of PDF combined with a ZIP-based package that is a forward-looking, competitive representation of PDF to address customer and competitive demands.
2. Support developers who what to leverage their XML tools and knowledge to create, manipulate, and extract information from PDF
3. Provide an XML document solution for organizations that have chosen to unify their infrastructure using XML as the base representation, and

4. Define and implement a representation of PDF information based on reusable XML components or subassemblies. These subassemblies represent self-contained pieces of document information that might be used in a variety of contexts including contexts not involving PDF documents.

There are several other important objectives for Mars:

5. Represent page content using a standard XML format. This format is SVG, a W3C "recommendation".
6. It should be possible to round-trip PDF extension data. This means that dictionaries in a PDFC file that are not defined in the PDF spec should be able to be converted to the Mars XML format, and that XML should be able to reproduce the extension dictionaries when the Mars containing them is converted back to a PDFC file.
7. Represent images, fonts, and color information as separate packaged files using standard formats such as JPEG, JPEG 2000, PNG, OpenType(R), and ICC.

It is expected that there would be several kinds of operations applied to Mars documents. They include

- Creation: Mars lowers the bar for creation of PDF documents. A variety of XML tools can be applied to help in the task, and dealing with COS syntax and COS object relationships is not required.
- Manipulation of auxiliary content: Mars makes the creation and manipulation of the non-content parts of the document much easier. Non-content parts include things like annotations, bookmarks, JavaScript™, fonts, metadata, external references, specialized processing data, and attachments.
- Document assembly and disassembly: Mars is designed to make page-level assembly of documents easy.
- Page content creation and manipulation: Mars also supports manipulation of page content which is based on SVG, a W3C graphics standard.

3 Mars Design Principles

There are a number of specific design principles on which Mars is based. Each is discussed in a section below.

3.1 Break a Monolithic Document into Pieces for Efficient Storage and Processing

PDF documents contain a number of different kinds of information including page contents, images, fonts, color definitions, for data, scripts, annotations, font information, and named destinations. Not all of this information is required to display parts of the document. PDFC documents are processed by reading only the required COS objects, resulting in quite good performance on accessing large documents. Mars documents should be structured so that in is not necessary to read all of the files in the Zip package into memory ahead of time. It should, in general, be possible to read in only the information required to perform a particular operation.

Breaking the document into separate files also provides a mechanism for efficiently accessing parts of the document. Each file represents a separate entry point into the overall document that can be read independently of other content.

3.2 Maximize Compatibility and Consistency With PDF

PDF was designed as a modular document format with separation of objects from different pages to allow optimal drawing time for individual pages. As a consequence, the structure of the content of PDFC is a good starting design for a document organization in XML. As PDF was defined years before XML, the needs of developers manipulating content in XML were not taken into account. Consequently, some deviations from the design of PDF are necessary. We still can greatly leverage the design of PDF and do not need to design a new format from scratch.

The Mars design is as consistent with the PDF specification as possible. Objects are broken down in a similar way and lexical conventions match PDFC to the greatest extent possible. Where possible, constant value names and tag names match the corresponding names used in PDFC.

3.3 Support Round-Trip

There are many workflows involving conversions between Mars format and PDFC (the current COS-based PDF format). We also want to encourage a vari-

ety of uses that we may not have envisioned. Some general classes of workflows include:

- Creation of Mars using XML tools, then convert to PDFC for distribution and processing
- Conversion of PDFC to Mars for use as a template; modification of the Mars template using XML tools; then convert back to PDFC for distribution and processing
- Conversion of PDFC to Mars for archiving, then later opening of Mars or conversion back to PDFC.

Being able to convert back and forth between PDFC and Mars is a common requirement. In most cases, we must round-trip convert between Mars and PDFC representations, and each representation should be able to carry extensions added to the other.

It is not necessary or required to maintain binary equivalence when converting between formats. It is not a requirement for digital signature to be preserved across a format conversion as such conversions are typically considered "changing all the bytes" in the file.

3.4 Maximize Independence of XML Elements and Components

Mars elements are designed to be independent and able to stand alone. This makes it easier to add, delete, rearrange, and reuse elements. For example, to add an annotation to page 3 of a document you should be able to stick an <Annotation> element into the document and associate it with page 3 and be done with it. But if your annotation requires special fonts you could expect to have to add a <Font> element somewhere as well. Similarly, to add, remove, or re-arrange pages, you can simply add, remove, or re-arrange <Page> elements.

To make elements more independent and reusable, the fine-grained sharing that is common in PDFC is avoided. When converting from PDFC to Mars format, shared PDF objects may be changed to be unshared in Mars by creating multiple copies of the objects. Large objects such as fonts and images are not duplicated and can remain as shared objects.

3.5 Self-Documenting

Consistent with developer use where developers may not be or want to be PDF experts, the XML in Mars is as self-documenting as possible. Tag names are descriptive of their content and obscure encodings are avoided.

Tag names, while consistent as possible with PDFC, are mnemonic. Some PDFC tags are very terse (one or two letters); these are not used in Mars.

The XML design should exhibit good esthetics of XML to the greatest extent possible. XML experts looking at Mars should say "yes, this is reasonable, useful XML".

3.6 Not an Internal Representation for Acrobat

Mars is not intended as the internal data structure for Acrobat. Acrobat APIs would remain as they are and plug-in developers would continue to use the interfaces as they have.

Future toolkits could be created that worked with Mars as a native format.

3.7 Not a Flowable Representation

Mars is an XML representation that matches PDF functionality. It is not a flowable, dynamic document format. It is isomorphic to PDF in information content and function.

3.8 Differences from PDFC

PDF documents consist of a number of different structures each serving a different purpose. The overall structure of PDFC is quite sufficient for use in Mars with a few exceptions. The basic design of Mars is to follow the structure of PDFC as closely as practical. There are three main areas of deviation

1. Optimized random access. There are many structures present in PDFC to provide optimized access to particular objects. This is useful especially in large documents. In XML, however, the general model is to read and parse the en-

tire XML file. Random access is achieved by breaking the overall document into separate XML files which are then packaged using Zip in a single file. The individual files need be read only when needed.

2. Binary encodings. For size and sometimes other reasons, structures in PDF may be encoded in binary or compressed. Since XML requires that binary content be encoded in printable text, binary or compressed structures are not compatible with an efficient XML representation. In addition, XML objectives include being readable and self-documenting to the extent practical which, again, mandates a more verbose, uncompressed textual encoding of information. Compression is achieved at the packaging level where both XML and other kinds of data are represented and compressed.

3. Cross-linked structures. XML data representations are easier to create and manipulate if they are independent and stand-alone. That is, all information relevant to a particular structure resides in the element that represents the structure. Sharing of objects and distribution of information across many elements (and its resulting pointer references) make it harder to reuse XML component representations and harder to add or delete objects. Therefore, to the greatest extent practical, Mars components are designed to be independent and sharing and cross-linkages are minimized. It is still necessary to share large objects such as images and fonts to maintain comparable file size.

Those familiar with PDF should have no trouble recognizing the various XML component representations that make up a Mars file.

3.9 Use Caches Instead of Complicating the XML

There are some places where, for performance reasons, information needs to be organized to optimize search rather than for developer convenience. This is required to support high-performance display and processing of documents for the end-user.

The design approach to address such file optimizations is to “cache” properly organized and formatted information in additional files stored within the document package. This avoids compromising the ease of document creation and manipulation achieved by keeping the XML elements independent and organized for ease of creation and manipulation.

The caches that optimize information access can be created in post processing steps or during content creation. The caches are potentially obsolesced when the document structure is changed; caches are recreated or updated in this case using a well-defined procedure. Each cache is self-describing so that document modification tools can update caches without having to understand what is being cached.

3.10 Support HTML-Like Document Linkage

The construction of HTML documents is a familiar process to many developers and there are tools available to assist in the management of linkages between document components. Mars is designed to have components of the document reference each other in a manner similar to HTML where relative URI links and fragment identifiers can be used to link between well-defined points in Mars documents.

3.11 Support Digital Signature Workflows

Increasing use of security features requires that Mars files support the ability to manage and represent signatures and to perform some operations without invalidating digital signatures.

This requires, at a minimum, that it be possible for form fill-in and addition of markup annotations to be done without invalidating signatures. Addition of document metadata is another modification that should be possible without invalidating signatures. A mechanism to implicitly associate annotations and metadata was defined to support these requirements.

3.12 Support for Obsolete or Deprecated PDF Features

It is desirable but not a requirement to be able to convert Mars format documents into version of PDFC older than PDF 1.7. Mars will not support the ability to represent and round-trip every feature of every version of PDF. In particular:

1. Deprecated PDF features may not be supported in Mars at all. Example: Pass-through PostScript(R).
2. If there are several ways to do the same thing in PDFC, only one of them (presumably the most "modern" incarnation of the feature) needs to survive in Mars. Examples: Type 1 and other legacy font formats are converted to OpenType. Several PDF color space families are converted to ICC profiles. NamedDestination dictionary converts to Mars named destinations, but would convert back to a named destination name tree.
3. If a PDFC is round-tripped to Mars and back to PDFC, the PDFC constructs that appear will be the result of this conversion; they may be quite different from what was in the original PDFC. This is explicitly deemed to be OK, so long as the appearance and behavior are preserved.
4. For behavior that is not defined by the PDF specification but by the Acrobat implementation, there is some flexibility to make changes if necessary. Example: Font substitution. (PDF doesn't prescribe how font substitution will be done, though it does specify the contents of the font descriptor that makes substitution possible.)

3.13 XML Conventions and Style

This section describes conventions used in Mars XML.

3.13.1 Tag and Attribute Names

With few exceptions, all tag and attribute names match the style of PDF which is to CapitalizeTheFirstLetterOfEachWord. All Mars-defined content is in the same namespace and the capitalization conventions apply to all names used in that namespace, regardless of where they appear.

Exceptions are “meta” attributes that describe character encoding and inter-object references, and a few cases where PDF uses lowercase.

3.13.2 Meta Attributes

Some attributes are used to describe file interconnection or character encoding. These attributes are connected using underscore and are written in lowercase. For example:

TABLE 1 Meta Attributes and Their Use

	EXAMPLE	COMMENT
src	<Bookmarks src=”file.xml”/>	File connectivity: lowercase src. src is always a file reference within the document package.
ref	<Screen ref=”/page/23/pg.can#b23”/>	Reference to specific XML content. ref is always a URI reference within the document package.
attr_ref	<Bead Page_ref=”backbone.xml#p23 ...>	Reference to specific XML content used in context where plain “ref=” wouldn’t be clear or when more than one ref from the element are required.
_enc	<OSName _enc=”Base64”>	Internal PDF encoding of string value. _enc always refers to element content encoding inside the PDF.
attr_enc	<OPI20 Comments=”28dj4” Comments_enc=”Base64”>	Internal PDF encoding of attribute string value. _enc is a suffix on the attribute name.
type	<Special type=”integer” Value=”23”/>	Identifies PDFC object type for custom extensions.
ref_type	<Private ref=”/page/23/pg.svg/ann#d12” ref_type=”array”/>	Identifies PDFC object type for references where the schema does not identify the referenced object’s type.

3.13.3 Reference Format

In general, references use the same tag as the object they are referring to. Thus, the reference serves as a stub for the actual object.

For example:

```
<ExData>
    <View ref=”#view23”/>
    ...
</ExData>

as a reference to

<View id=”view23” ....> ... </View>
```

In cases where the intervening element is not desired, a more specific ref attribute can be defined:

```
<ExData View_ref="#view23" ...>
```

The same approach is applied to references that use names or other conventions:

```
<Field Pathname="a/b/c"/>

as a reference to

<Field Name="c" ...> ...
```

3.13.4 Plural Structures

In places where one or more of the same kind of object can appear, the following XML structure is used:

```
<Pages>
    <Page .../>
    <Page ...>
    ...
</Pages>
```

The general idea is to enclose the list in a plural tag for the items in the list. This helps eliminate the need to proscribe the order of elements. A list with one entry can optionally eliminate the plural element. Look carefully at the schema to see if this is allowed.

3.13.5 Use Attributes Whenever Possible

Attributes offer a more compact representation and are generally easier to process because they are parsed along with the opening of the element which contains them.

In Mars, attributes are used whenever possible to represent scalar values. Elements may be used in a number of circumstances including, but not limited to,

- The value may contain structure and uses nested tags
- A collection of attributes makes more sense as a group
- Attribute naming conflicts need to be resolved

- The attribute value might be really large and/or its value might want to be represented with CDATA

Attributes are also used in limited cases for structured values. The guidelines for such use are:

- The data types should be homogeneous, most often lists of numbers
- If the values represent different kinds of information, the length should be limited to 6 items. If the values represent a list of the same kinds of information, e.g., coordinates, then the list can be longer.
- Parsing the attribute value should be very simple when these guidelines are followed: simple tokenization and conversion.

3.13.6 Members of Class

In PDF, there are a number of object classes such as Annotations, Form Fields, or Actions. In the XML representing objects of these classes, the XML tag is the specific instance of the class and not the class itself. Generally, the class is only evident in the schema, not in the XML data itself. For example, the annotations are represented by elements such as <Text>, <Line>, and <Underline>. In the schema, there is an definition markup_annotation_dictionary that represents the class of annotations. There can be places where a reference to a class member uses an element such as <Annotation ref="..."/>.

4 Mars Document Structure

4.1 Mars Document Overview

A Mars document consists of pages with graphic content such as text and images, document navigation information such as bookmarks and articles, markup information representing reviewer comments (for example), and other kinds of information. All of this is packaged together into a ZIP package. The files in the package make up the complete content of the document.

This section gives a brief overview of various features of Mars documents and how they are represented.

Document Root

The root of the document is in the file /backbone.xml. This file contains some global document information as well as some minimal information about each page. Form field information is also in the backbone as well as references to bookmarks, web capture, and other application information. The backbone file must always be read so it is kept as small as possible.

Page Contents

In the document backbone, the pages are represented by a series of XML <Page> elements. Each of these points to a separate page information file. The graphic page content itself is represented using SVG in a third file that is referenced from the page information file. Pages can also include information about the logical structure of the page contents. Some structure information is included on the page interspersed with the SVG itself. Additional structure information, named destination information, resources, and annotations are stored in separate files associated with the page to which they apply.

Fonts

Fonts can be embedded in the document itself or externally referenced. For embedded fonts, an encrypted open type format font file is referenced directly from the SVG content that uses it. Non-embedded fonts reference a font descriptor

file which provides information that enables finding the best font to use given the set of locally installed fonts. The font descriptors are represented in XML and can be shared across pages in a Mars document.

Annotations

Annotations are placed in separate files, grouped by page. For each page, there can be two annotation files, one containing markup annotations such as sticky-notes and text comments, and another containing content-oriented annotations such as form field widgets and hyperlinks.

Bookmarks

The bookmarks (outline items in PDF) are represented in XML in a separate package file that is referenced from the document backbone file. Bookmarks can simply be a table-of-contents for the document or can have more complex behavior.

Metadata

Document level metadata is stored in a separate file in the META-INF directory and contains the XMP representation of the metadata. Metadata associated with other package objects such as pages, fonts, or images, appear in separate files related to the files that they describe.

Object-level metadata which describes page-level graphic objects is represented using XML markup within the SVG page contents file.

Logical Structure and Tagged PDF

In PDF, document structure information is represented as a tree whose leaves are the content items themselves. Interior nodes of the tree represent chapters, sections, tables, figures, and so on. In Mars, rather than having a single document-wide tree, structure information for a particular page is represented as markup that is part of that page and in a file associated with that page, making it easier to create pages and move pages between files.

The role map and class map, which map custom structure tags to standard role tags and provide default structure class attributes, respectively, are stored as part of the document backbone.

Information indicating which pages have the structure information for which interior logical structure tree nodes is stored in a cache file to facilitate lookup during tree traversal. For more detail, see section 16, “Marked Content & Logical Structure”.

Web Capture

PDF documents can represent captured images of web pages. Some web capture information is stored in the document backbone. This information generally provides the context in which the PDF was created from web pages and allows the PDF to be updated from the same web pages. The URL and ID maps that are part of this information are stored in separate package files that are referenced from the document backbone.

Optional Content

Optional content is a mechanism for specially marked content to be displayed or hidden under user and document control. Optional content groups, configurations, and so on are stored in the document backbone. Optional content information that is part of the page content is stored as markup on the SVG page.

Multimedia

Media selection, rendition information, player information, and so on are stored in the document backbone. Movie annotations are part of the content annotations stored with its associated page (in the separate content annotations file). Sound annotations are markup annotations also stored with its associated page (in the markup annotations file). The annotations themselves refer to the media files which are within the document package.

File Attachments

If files are embedded within the document, the document backbone contains a reference to a file which lists all of the embedded files. File attachment annota-

tions are considered markup annotations stored with other annotations associated with a page. The attached file itself is stored as a separate file within the document package (in the /file directory).

Pages

The basic list of pages is stored within the document backbone. The information about the page itself is broken into up to 8 or more files: the page information file, the page content file, the page structure file, the named destinations file, the content annotations file, the markup annotations file, page-level metadata (xmp) file, the page resources file, and any images, fonts, font descriptors, functions, shaders, patterns, color profiles, and others. In addition, the page content itself can be broken into multiple files using the SVG symbol mechanism.

Page Resources

In some cases, SVG content will refer to resources in other files using a URI. These external resources can be shared by multiple pages. Examples of resources include images, color spaces, shaders, fonts, and font descriptors.

Color Spaces

SVG directly supports ICC color profiles. The ICC profiles themselves are stored as separate package files. A number of PDF-defined color spaces are supported to represent color features not covered by ICC profiles. These are represented as markup within SVG files (in the pdf namespace) or within separate resource files.

Named Destinations

Named destinations is a PDF mechanism to refer to a location within a document by name. (It is an internal mechanism, not visible to readers of the document.) In Mars, the named destinations are grouped by page and stored in a file associated with the page to which they refer. A separate cache file maps the names to the page file where the details of the destination are stored, enabling fast lookup.

Form Fields

Form fields in PDF consist of three parts: some base information, form fields, and form field Widgets. Base information about the form is in the AcroForm element within the document backbone. In addition to general information about the form, this element includes elements representing each form field. The form field widgets, representing the visible manifestations of the fields that are displayed on the page(s) of the document, are stored in the content annotations file associated with each page.

The field elements refer to the widget annotation elements using an uri reference scheme. This crosses files as the fields are in the backbone and the widgets are in annotation files referenced by the pages on which they appear.

The widgets are likely to refer to appearance streams which are SVG content files that describe how to render them. These SVG files are separate package objects.

XFA Forms

XFA forms include more XML capabilities and a richer programming model than basic AcroForms. In addition to the information present for a non-XFA form, XFA forms have one or more additional files. The individual files are referenced by the backbone.

4.2 Mars Package File Format

A Mars document is essentially a ZIP file that follows a few additional requirements. The details of the file format are discussed in Appendix E “Universal Container Format” on page 167.

4.3 Mars File Organization

A Mars document consists of a number of separate files. A single root file, called the “backbone.xml” describe the overall document and points to most of the other pieces.

Most files can have any name and be in almost any location in the package as long as the URI references to the file use the proper name. The next section covers files that must be in specific locations with specific names.

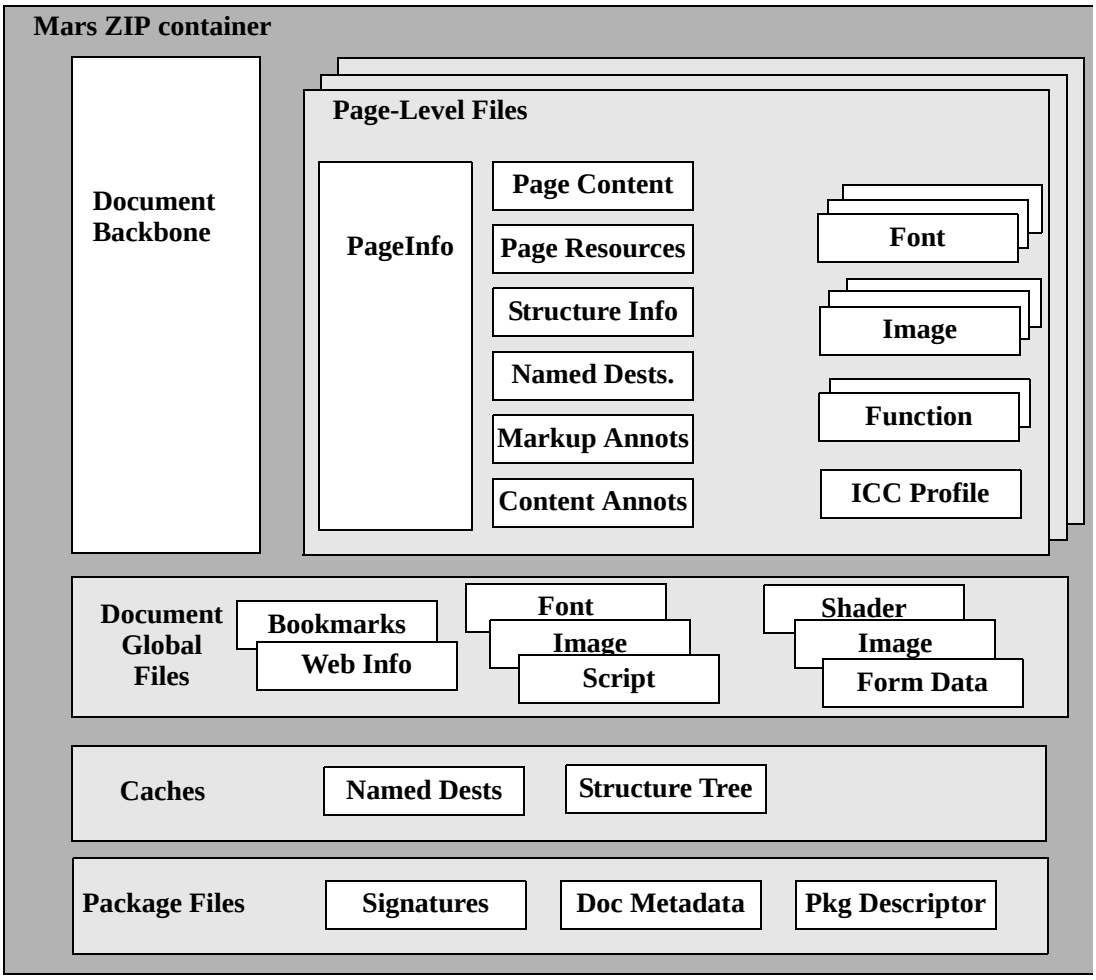


FIGURE .1 *Logical Structure of a Mars document.*

4.3.2 File Organization Rules

Some files must be specifically named and exist in a prescribed directory structure within the package. Other files are organized using certain conventions but their naming and place in the package directory structure is not mandated by Mars.

File Names

All file names within a Mars document must conform to restrictions defined by ZIP and the Container Format Specification (see "Universal Container Format" on page 167).

Specific Files and Directories

The root directory must contain

- the file "mimetype" which must be physically the first file in the package. It's contents must be the Mars mime type which is "**application/vnd.adobe.x-mars**". This is a "code name" mime type; the final mime type will be chosen when the name for the format is finalized.
- the file "backbone.xml" which is the root file of the document, describes the overall document, and references other files (though not all) which comprise the document.
- the directory "META-INF" (see below)
- the file "/form/form_data.xfdf". This file is optional and contains the form data for non-XML forms (AcroForms). This name is reserved for that purpose only.
- the optional directory /cache, if it exists, may contain only cache files.
- the optional directory /file. If there are any embedded files, they must be located in this directory.

META-INF Directory

The META-INF directory is reserved for packaging-specific files and may contain only the following files:

TABLE 2 META-INF Package Files

FILE	REQUIRED?	CONTENTS	REFERENCE
META-INF/container.xml	Yes	Basic package information and relationship class information	CF (see E)
META-INF/metadata.xml	No	Document level metadata in XMP format	CF (see E), XMP
META-INF/signatures.xml	No	Digital signatures on document contents	CF (see E)
META-INF/encryption.xml	No	Encryption information on package contents.	CF (see E)
META-INF/rights.xml	No	Special document capabilities	Mars

4.3.3 File Organization Conventions

Although most of the files in a Mars package can be located anywhere in the directory structure, the following conventions are recommended.

TABLE 3 Conventions for Mars File Placement

DIRECTORY PATH	CONTENTS
/page/<i> where <i> is the zero-based page number or a page name	Page content files for page i would go in /page/i/. Note that due to page insertion, deletion, or rearrangement, clients should not assume that files in /page/7 (for example) necessarily correspond to the eighth page. Also, directories like /page/5a might exist to cover cases for a page inserted between 5 and 6. A useful convention is to number pages with a fixed number of digits in the filename (e.g., 0001, 0002, etc.). This can make working with the files by hand easier because the sort order will keep the pages in order. Making the pathnames excessively long should be avoided however as it tends to increase file size unnecessarily. Files known to be used exclusively by a particular page should be placed in the directory for that page.
/font	Fonts, cmaps, and other font-related files.

TABLE 3 Conventions for Mars File Placement

DIRECTORY PATH	CONTENTS
/color	Color definitions and look-up tables.
/image	images used in the document.
/obj	shared page fragments
/res	resources used by graphic content, generally functions, patterns, shaders, etc.
/misc	Other kinds of files that are used by objects within the document that don't fall into one of the other categories.
/script	JavaScript script files.
/annot	additional content required by annots
/form	Form related files
/media	Sound and movie annotation files.
/web	Data files used by web-capture.

4.4 Mars File Types

Each of these types is described in more detail in the referenced section or standard.

TABLE 4 Mars File Type Summary

FILE TYPE	DESCRIPTION	TYPICAL SUFFIX / MIME	STANDARD	SECTION
Backbone	Overall document structure and description. Filename must be "backbone.xml"	.xml	Mars	5
Bookmarks	Bookmarks	.xml	Mars	5.4
Web Capture ID List	Information about each web element represented in the PDF file.	.xml	Mars	
Web Capture URL List	Information about each URL visited during capture of a web site	.xml	Mars	
Application Data	Page-piece data and output intents	.xml	Mars	5.8

TABLE 4 Mars File Type Summary

FILE TYPE	DESCRIPTION	TYPICAL SUFFIX / MIME	STANDARD	SECTION
Annotation Appearances	Default annotation appearance information	.xml	Mars	5.12
Scripts	Javascripts and references	.xml	Mars	5.13
Embedded File Directory	List of embedded files	.xml	Mars	5.14
Articles	Article threads	.xml	Mars	5.11
Page content	Graphic, text, and structure information for each page. Also used for page fragments.	.svg	W3C SVG	6.2
Page Information	Lists basic information about a page and refers to additional files containing more details about the page.	.xml	Mars	6.1
Page Resources	Resources used on a given page(s)	.xml	Mars	
Page Structure	Information about the structure parent nodes of marked content that is on a given page.	.xml	Mars	6.3
Page Destinations	Named destinations that refer to a given page.	.xml	Mars	6.4
Font descriptor	Supplementary information about a font	.fd	Mars	10.3
Font	A font or font subset	.otf _sfnt	OTF	10
Image	jpeg compressed image	.jpg		11
Image	jpeg2000 image	.jp2		11
Image	jbig2 image	.jb2		11
Image	png image	.png		11
Sampled Function	PDF Type 0 function	.f0	PDF	PDF 1.6 S3.9.1
Postscript Function	PDF Type 4 function	.ps	PDF	PDF 1.6 S3.9.4
Pattern Data	PDF Type 1 pattern	.svg	W3C SVG	6.2
Shader Data	PDF Type 4-7 shading data	.sd	PDF	PDF 1.6 S4.6.3

TABLE 4 Mars File Type Summary

FILE TYPE	DESCRIPTION	TYPICAL SUFFIX / MIME	STANDARD	SECTION
ICC Profile	ICC Color definitions	.icc	ICC	ICC.1:2004-04
Markup Annotation	XFDF format document annotations	.ann	Mars	13
Content Annotation	Form fields, links, movies, 3D, etc.	.can	Mars	13
Metadata	Document or component metadata	.xmp	XMP	15
Postscript Object	Postscript	.ps		
Halftone Colorant	Halftone Colorant information for type 5 halftones	.htc	PDF	
Rich text	Rich text content for an annotation	.xml		
Javascript	Javascript source code	.ajs	ECMAScript	
XML Form Template	XFA template file	.xft		9.1
Sound	Sound annotation	e.g., .wav		various formats supported - See Acrobat
Web Capture Post data	Web Capture post data used when content was originally captured	.txt		
Indexed Colorspace LUT	Lookup table for indexed color space	.lut	PDF	PDF 1.6 S4.5.5
3D Model	U3D Standard format	.u3d	ECMA-363	PDF 1.7 S9.5
3D Model	PRC format	.prc		PDF 1.7 S9.5

4.5 Mars Component References

There are three kinds of component references: explicit resource references, implicit resource associations, and explicit element references.

4.5.1 Explicit Resource References

Explicit resource references are used when there is a specific need to reference a file within the document package. References to fonts and images are good examples of explicit resource references.

Example: <Page src="/page/5/info.xml" x1="0" y1="0" x2="612" y2="792"/>

At the reference point, an element with a “src” attribute names the component via a URI. The URI must be either relative or root-based but cannot specify a different scheme and must refer to an object within the same package as the referencer. When more than one reference must be placed on the same element, an attribute other than “src” is used.

4.5.2 Implicit Resource Associations

Implicit resources associations are used when there is not a specific pointer from one Mars resource to a referenced resource. The existence of the resource under a particular name is used to form the association between the two resources. For example, annotations and metadata can be connected to particular Mars document resources by their existence in the Mars package under a specific name.

Example: /page/3/im_123.jpeg and /page/3/im_123.jpeg.xmp would be related using the metadata association rule even though there is no explicit naming of the .xmp file in the .jpeg file or anywhere else.

Classes of implicit associations based on filename are defined in the META-INF/container.xml file. These relationships define implicit references between pairs of package-level files. Implicit relationships are used because they allow addition or removal of information without modifying either the referencer or the referenced component. This enables changes such as adding annotations or metadata that do not affect digital signatures on document content. See Section E.3.5.1.2 “Relationships (Optional)” on page 182

4.5.3 Explicit Element References

A URI that can include a file reference and an id reference is used to reference a specific XML element in one resource from another resource. The URI reference includes the name of the referenced resource much like an explicit resource reference. The URI also includes an identifier that identifies a specific element via its ID attribute.

Example: <Widget ref=”content.can#W3452”/> relative URI
<Widget ref=”/page/23/content.can#W3453”/> fully-qualified URI
<Widget ref=”#W3451”/> local reference in same XML file as the reference

In some cases, references are made within a single file using a simple name. There are specific rules for the given context stating how the reference is to be resolved. For example, in an SVG file, a color reference such as

fill="rgb(246,177,153) icc-color(cs-1,80,20,20)"

references a color space definition named “cs-1”:

<svg:color-profile name=”cs-1” ... />

4.5.4 External Objects References

PDF supports the concept of referencing objects outside the document itself. There are two main cases of this: links to external web pages (from URI actions, form submit, and web capture) and references to external files (usually for large images or fonts). For security reasons, the latter references are disabled by default in viewing applications.

There is no general mechanism in Mars to support external web links. Web links are supported by specific PDF-defined element/attribute combinations. Only those element/attribute combinations will be interpreted as web links.

External files have a more general representation in PDF and hence Mars. If a stream includes a file_spec_dictionary and no stream data (specified with the src attribute), then the stream contents is considered an external reference and the file_spec_dictionary is used to determine the external filename.

4.6 Character and Text Encoding

Character and text encoding issues that arise when converting between Mars and PDFC formats are discussed in Section D “Character Encoding” on page 151.

4.7 Preservation Rules for Non-Mars Files in a Mars Package

Files defined in this specification and referenced directly or indirectly from the backbone.xml or cache files or referencable via relationships defined in META-INF/container.xml element <relationships> comprise a Mars document. Other files that may be present in the package define extension data. When tools operate on Mars documents, any file located in the root directory or the /file directory

must be preserved. Extension files in other directories in the package need not be preserved.

Files referenced from Mars-defined XML using src= or ref= attributes are considered part of the Mars document. This includes stream extension references in extension data located within Mars-defined XML. See Section 21 “Foreign XML” on page 131 and Table 29 on page 132. It does not include references that appear within extension files which might happen to contain XML. Thus, if XML in a defined extension area includes a ref= or src= reference to a file, that file is considered part of the Mars document and must be preserved by tools operating on Mars format files.

5 Mars Backbone File

5.1 Namespace

All tags in Mars are part of the Mars namespace defined by

http://ns.adobe.com/pdf/2006

If a namespace prefix is to be used it should conventionally be “pdf:”, but for file size reasons, the contents of Mars XML uses this namespace as the default. Foreign data should be tagged with explicit namespace declarations.

In SVG files, svg should be the default namespace. There, Mars marked content tags are in the Mars namespace and may need explicit namespace prefixes.

5.2 Top-Level Element

The top level element is a Mars file is <PDF>. This element must have an attribute “PDFVersion” that identifies the PDF format version, and a “Version” attribute that identifies the Mars schema version, and a “Class” attribute that identifies the document class (think of this as a really-major version number; failure to match implies that the file can’t be sufficiently recognized to be processed). The value of “Version” is of the form “major.minor.micro” where each of major, minor, and micro are numbers. “1.1.9” is considered earlier than “1.1.10”.

The contents of this element, referred to as the “Mars backbone”, correspond to information that would appear in the catalog dictionary of a PDF.

5.3 Mars XML Backbone

All Mars consumers and producers must support UTF-8 character encoding.

The “root” of the Mars document is the Mars backbone in the file “/backbone.xml”. This file contains direct references to other files that comprise the Mars document. Files referenced from the backbone may in turn reference other files that are part of the document but are not directly referenced from the backbone. In addition, files may be part of the document through implicit resource associations (see 4.5.2, “Implicit Resource Associations”).

The core XML structure of a Mars file includes the basic content of the catalog dictionary in a PDF file. As with most Mars XML, the child elements can appear in any order.

Example 1 Sample Mars Backbone

```
<?xml version=”1.0” encoding=”UTF-8”?>
<PDF PDFVersion=”1.6” Version=”0.8.0” PageMode=”UseOutlines”
PageLayout=”TwoColumnRight” Class=”04-18-07” xmlns=”http://ns.adobe.com/pdf/2006”
DocumentID=”6srRqThA2GMw8NkJpVoSyA==” InstanceID=”LM0HthVCukGq9r/2dbgYYg==”>
    <Marked UserProperties=”true”/>
    <ViewerPreferences CenterWindow=”true” DisplayDocTitle=”true” HideToolbar=”false”/>
    <AnnotationAppearances src=”/default_appearances.xml”/>
    <Bookmarks src=”/bookmarks.xml”/>
    <Metadata src=”/META-INF/metadata.xml”/>
    <AcroForm>
        <Fields>
            <Button Name=”Check Box3” Widget_ref=”/page/0/pg.can#0”
Widget_ref_type=”dictionary”>
                <DefaultAppearance Font=”ZaDb” TextSize=”0” FillColorSpace=”Gray” FillColor=”0”/
>
            </Button>
            <Button Name=”Check Box5” Widget_ref=”/page/3/pg.can#10”
Widget_ref_type=”dictionary”>
                <DefaultAppearance Font=”ZaDb” TextSize=”0” FillColorSpace=”Gray” FillColor=”0”/
>
            </Button>
            <TextField Name=”Text9” Widget_ref=”/page/3/pg.can#3” Widget_ref_type=”dictionary”>
                <DefaultAppearance Font=”Garamond” TextSize=”12” FillColorSpace=”Gray”
FillColor=”0”/>
            </TextField>
        </Fields>
        <DefaultAppearance Font=”Helv” TextSize=”0” FillColorSpace=”Gray” FillColor=”0”/>
```

```
    <Resources/>
</AcroForm>
<Pages>
    <Page src="/page/0/info.xml" x1="0" y1="0" x2="612" y2="792" ID="7"/>
    <Page src="/page/1/info.xml" x1="0" y1="0" x2="612" y2="792" ID="0"/>
    <Page src="/page/2/info.xml" x1="0" y1="0" x2="432" y2="288"/>
    <Page src="/page/3/info.xml" x1="0" y1="0" x2="612" y2="792"/>
    <Page src="/page/4/info.xml" x1="0" y1="0" x2="612" y2="792"/>
    <Page src="/page/5/info.xml" x1="0" y1="0" x2="612" y2="792"/>
    <Page src="/page/6/info.xml" x1="0" y1="0" x2="1584" y2="2448" ID="55"/>
    <Page src="/page/7/info.xml" x1="0" y1="0" x2="1584" y2="2448" ID="31"/>
    <Page src="/page/8/info.xml" x1="0" y1="0" x2="612" y2="792" ID="79"/>
</Pages>
<OCProperties>
    <Default>
        <ExclusiveGroups/>
        <ON>
            <Group ref="#1"/>
        </ON>
        <PresentationOrder>
            <Groups>
                <Label>File1.pdf</Label>
                <Groups>
                    <Label>NestedLayers.pdf</Label>
                    <Group ref="#2"/>
                    <Group ref="#4"/>
                </Groups>
                <Group ref="#78"/>
            </Groups>
        </PresentationOrder>
    </Default>
    <Groups>
        <Group Name="Watermark" ID="1">
            <Usage>
                <Export ExportState="ON"/>
                <PageElement Subtype="FG"/>
                <View ViewState="ON"/>
                <Print PrintState="ON"/>
            </Usage>
        </Group>
        <Group Name="Black Text and Green Snow" ID="2"/>
        <Group Name="Mountains and Image" ID="3"/>
        <Group Name="Starburst" ID="4"/>
        <Group Name="Watermark" ID="5">
            <Usage>
                <Print PrintState="ON"/>
            </Usage>
        </Group>
    </Groups>
</OCProperties>
<AfterOpen>
    <GoTo>
```

```
            <Dest>
                <XYZ Left=”-32768” Top=”33560” Zoom=”0.75” Page_ref=”#7”/>
            </Dest>
        </GoTo>
    </AfterOpen>
    <Threads src=”/articles.xml”/>
    <StructureTypes>
        <RoleMap>
            <Role Name=”Sheet.571” MapTo=”Figure”/>
            <Role Name=”Sheet.268” MapTo=”Figure”/>
        </RoleMap>
        <ClassMap>
            <Class Name=”Vertical”>
                <Attribute Owner=”UserProperties”>
                    <UserProperties>
                        <UserProperty FormattedValue=”ï»¿90 deg.” Name=”ï»¿Angle” Value=”90”
type=”number”/>
                        <UserProperty Hidden=”true” Name=”ï»¿Extension Lines” Value=”ï»¿Both”
Value_enc=”UTF-16” type=”string”/>
                        <UserProperty Name=”ï»¿Precision” Value=”ï»¿0.00” Value_enc=”UTF-16”
type=”string”/>
                    </UserProperties>
                </Attribute>
            </Class>
        </ClassMap>
    </StructureTypes>
</PDF>
```

TABLE 5 Mars Backbone Schema Overview

SCHEMA	NOTES
root = element PDF { catalog_dictionary }	Top level tag <PDF>
catalog_dictionary = attribute Version { string }	
& element Pages { page_tree_dictionary }	
& element PageLabels { page_label_dictionary_wrapper* }?	Lists Pages
& name_dictionary?	
& element ViewerPreferences { viewer_preferences_dictionary }?	
& attribute PageLayout { layout_type }?	
& attribute PageMode { pagemode_type }?	
& element Bookmarks { attribute src {string} }?	
& element Threads { attribute src {string} }?	
& element AfterOpen { action_or_destination }?	
& document_additional_actions_dictionary?	
& element URI { uri_base_dictionary }?	
& element AcroForm { acroform_dictionary }?	
& element Metadata { attribute src {string}}?	
& element StructureTypes { structure_tree_root_dictionary }?	
& element Marked { mark_information_dictionary }?	
& attribute Lang { pdf_text_string }?	
& element WWW { web_capture_information_dictionary }?	
& element OutputIntents { array_of_output_intent_dictionary }?	
& element ApplicationDatasets { attribute src {string} }?	
& element OCProperties { oc_properties_dictionary }?	
& element Legal { legal_dictionary }?	
& element Requirements { array_of_requirement_dictionaries }?	
& element Collection { collection_dictionary }?	
& attribute NeedsRendering { boolean }?	
& attribute PDFVersion { pdf_text_string }?	
& attribute Class { text }	
& attribute DocumentID { string }?	
& attribute InstanceID { string }?	
name_dictionary = element AnnotationAppearances { attribute src {string} }?	
& element JavaScripts { attribute src {string} }?	
& element Templates { template_page_name_tree* }?	
& element WebIds { attribute src {string} }?	
& element WebURLs { attribute src {string} }?	
& element EmbeddedFiles { attribute src {string} }?	
& element AlternatePresentations { alternate_presentations_name_tree* }?	
& element Renditions { renditions_name_tree* }?	

5.4 Bookmarks

Bookmarks, also called outline items, are an indexing structure that is displayed and processed by PDF viewers. Each bookmark has a title and an action to be performed when it is activated.

Bookmarks are in a separate file referenced from the Mars backbone. Note that bookmarks can reference page ordinals, the structure tree, or contain arbitrary actions.

Example 2 Bookmarks Example

in the backbone.xml file:

```
<PDF ...>
   <Bookmarks src=”bookmarks.xml”/>
   ...
</PDF>
```

in bookmarks.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<Bookmarks xmlns="http://ns.adobe.com/pdf/2006" Open="true">
    <Bookmark>
        <Action>
            <GoTo>
                <Dest>
                    <XYZ Left="72" Top="587" Zoom="" Page_ref="/backbone.xml#2"/>
                </Dest>
            </GoTo>
        </Action>
        <Title>Company Overview</Title>
    </Bookmark>
    <Bookmark>
        <Action>
            <URI URI="http://www.adobe.com"/>
        </Action>
        <Title>Visit Global On the Web</Title>
    </Bookmark>
    <Bookmark>
        <Action>
            <Sound>
                <Sound src="/media/sound_4209265555-6" SamplingRate="22050"/>
            </Sound>
        </Action>
        <Title>Audio Introduction</Title>
    </Bookmark>
</Bookmarks>
```

In most documents, each bookmark contains a single Goto action that displays a particular page and view of the page. Bookmarks can contain one or more actions, including JavaScript actions, that can have complex behavior.

5.5 Document Actions

The document actions specified at the document root level define JavaScript code to be executed before or after open, close, save, and print. The JavaScript code can be directly in the backbone XML or can be a separate package level file. See “Actions” on page 107 for more about actions.

5.6 Pages

Pages each appear as a separate set of files. Each page element includes a reference to the page information file and contains attributes that specify the page media box (the bounding box for page contents). UserUnit is also specified as part of the Page element.

The <Pages> element in the Mars backbone appears as:

```
<Pages>
    <Page src=”/page/0/info.xml” x1="0" y1="0" x2="612" y2="792"/>
    <Page src=”/page/1/info.xml” x1="0" y1="0" x2="612" y2="792"/>
    ...
</Pages>
```

5.7 Viewer Preferences

Viewer preferences in PDF control view settings when the file is opened and other Viewer behaviors. The <ViewerPreferences> element is a child of the PDF element in the Mars backbone.

Example 3: Viewer Preferences

```
<ViewerPreferences CenterWindow="true"
                   DisplayDocTitle="true"
                   HideToolbar=”false”
                   NonFullScreenPageMode=”UseOC”
/ >
```

5.8 Custom Application Information

The Page Piece dictionary in PDFC represents application-specific extension data that can be associated with the document, individual pages, and fragments of pages (form xobjects). The XML markup representing the application-specific information associated with the document is stored in a separate file referenced from the document backbone. The root element of this file is <ApplicationDatasets>.

The <ApplicationDatasets> element and its contained application-specific information can also appear in the page information file as part of the <Page> element.

The backbone refers to a separate file that contains the actual application datasets. The root element of this file is <ApplicationDatasets>. In other contexts, the <ApplicationDatasets> element and the contained application datasets themselves are inline.

The <ApplicationDatasets> element has the following format. Note how the externally-defined elements are flagged with the underlying type so that they can be consistently processed. Scalar and structured values can appear in private data.

Example 4 ApplicationDatasets markup

```
<ApplicationDatasets>
    <ApplicationData Owner=”Jones Formatter 1.2” LastModified=”20430626221926”>
        <Private type=”name” Value=”NoRecallSpellingHollars”/>
    </ApplicationData>
    <ApplicationData Owner=”Flex 1.1”  LastModified=”20030226221926”>
        <Private type=”dict”>
            <Server type=”string” Value=”Acroblab1”/>
            <Age type=”int” Value=”23”/>
        </Private>
    </ApplicationData>
<ApplicationDatasets>
```

5.9 Page Labels

The page_label_dictionary in PDF defines how pages numbers should be displayed in a viewer application. It does not affect visible page content. In PDFC,

the page labels are stored in a number tree, a balanced search tree. In Mars, a simple list of entries specifying page ranges and label formats is used.

Example 5 Page Labels Example

```
<PageLabels>
    <PageLabel Start="0" Style="Roman_Lowercase"></PageLabel>
    <PageLabel Start="22" Style="Numeric"></PageLabel>
</PageLabels>
```

5.10 Optional Content

Optional Content in PDF consists of a number of structures that describe sets of content items and labels to use in a user interface to enable them to be selectively displayed. Individual content items are grouped and tagged using marked content containers that wrap the actual page content items.

In Mars, the structures are represented in the document backbone and the content information is represented on the pages, wrapping relevant SVG page content. The backbone information is the <OCProperties> element.

5.11 Article Threads and Beads

Threads and beads define a mechanism to link together a series of rectangles in the document. This can give the user a means to navigate the document in a logical way assuming each series of rectangles represents a flow of logical content, such as magazine articles which are continued over a series of non-contiguous pages. This mechanism is not really used any more and was replaced with logical structure and tagged PDF.

Article thread information is all stored within a separate file referenced from the Mars document backbone. The root element of this file is <Threads>. A Thread consists of a series of Beads each of which is a rectangular region which points to the page on which it lies. There is no information on the page related to Threads and Beads.

5.12 Annotation Appearances

Default named annotation appearances are stored in a separate package level file that is referenced from the backbone file. The root element of the annotation appearances file is <AnnotationAppearances>. A reference from the backbone appears as:

<AnnotationAppearances src="annotation_appearances.xml"/>

Note that Annotation Appearances is used to help manage appearances by an application to avoid duplication. A Mars creation application does not have to include this element or the file to correctly use annotations or form fields.

5.13 Scripts

Named, document-level JavaScript actions are stored in a separate package level file that is referenced from the backbone file. The root element of the scripts file is <JavaScripts>.

5.14 Embedded Files

Embedded files are listed in an xml structure in a package level file. The root element of the string is <EmbeddedFiles>. The document backbone refers to this file. For further information on embedded files, see Section 8 “Embedded Files” on page 72.

<EmbeddedFiles src="/file/embedded_files.xml">

6 Page Files

The information for a page is spread across a number of files. Each of these files is described in a section of this chapter. Annotations also contain a graphic description of their appearance. Because of this, they share some graphic characteristics with pages.

6.1 Page Information File

The page information file contains basic information about the page. It must be read in order to know anything about the page. Information in the page information file includes:

1. A reference to the page content file.
2. A reference to the page named destination file, if any.
3. Page size and orientation information.

TABLE 6 Page Information File Schema Overview

page_file = element Page { page_dictionary } page_dictionary = attribute LastModified { date }? & attribute Rotate { "0" | "90" | "180" | "270" }?	
& element Contents { attribute src {string} } & element Annotations {attribute src {string}}? & element Destinations { attribute src { string } }? & element Structure { attribute src { string } }?	References to other files that comprise the page. Annotations refers to content annotations.
& element CropBox { rectangle }? & element BleedBox { rectangle }? & element TrimBox { rectangle }? & element ArtBox { rectangle }? & element BoxColorInfo { box_color_information_dictionary }?	Page size structures. (MediaBox size is in the document backbone.)

TABLE 6 Page Information File Schema Overview

& element Attributes { group_attributes_dictionary }? & element Thumbnail { image_dictionary }? & element ArticleBeads { array_of_bead_dictionary_reference }? & attribute Dur { number }? & element Trans { transition_dictionary }? & element OnPageOpen { action_dictionary }? & element OnPageClose { action_dictionary }? & element ApplicationDatasets { page_piece_dictionary }? & attribute WebCaptureId { pdf_base64_string }? & attribute PreferredZoom { number }? & element SeparationInfo { separation_dictionary }? & attribute Tabs { name }? & attribute Tabs_enc { token }? & attribute TemplateInstantiated { name }? & attribute TemplateInstantiated_enc { token }? & element PresSteps { navigation_node_dictionary }? & attribute UserUnit { number }? & element ViewPorts { viewport_dictionary_array }?	Other general page information

6.2 Graphic Content

Graphic content is represented in SVG format, a W3C standard for representing text and graphics. Details of the representation of graphics are covered in Section 7 “Graphic Content” on page 44.

6.3 Marked Element and Logical Structure Representation

The Marked-content facility in PDF allows arbitrary sequences of content to be grouped and “marked”. This marking mechanism allows applications or plug-ins to later identify or access the content and to associate metadata with it.

The structure tree in PDF provides a logical description of the document contents. It’s view is more along the lines of an HTML document with chapters, sections, paragraphs, tables, figures, articles, and so on. The structure tree contains nodes which typically describe chapters, sections, tables, and so on, and leaf nodes which typically describe content items such as paragraphs, table cells, headings, and so on. These leaf nodes contain marked-content references to con-

tent on the page. In the case that a marked content element on the page is used for logical structure, it must contain an ID attribute.

In a Mars document, each page can have an associated structure file. This page structure file lists structure information for the nodes of the structure tree that are parents of leaf nodes that reference content on the page. The SVG page content itself contains the marked content elements for the leaf nodes of the structure tree. Each page includes structure information for the structure tree elements that span from the structure tree root to the leaves that appear on the page. Thus, each logical page contains complete information for the portion of the structure tree needed to describe content on it (via a combination of the page content file and the page structure file).

Each structure node in the page structure file identifies its location in the structure tree using a notation which effectively provides a path through the tree from the root to the structure node itself.

All structure information is distributed onto individual pages which contain the content items ultimately referred to by the structure tree.

Marked content containers are represented as follows:

TABLE 7 Structure Information in Pages

PDF CONTAINER TYPE	TYPICAL MARS REPRESENTATION
MP - Marked Point	<pdf:Point pdf:Mark="name"/>
DP - Marked Point with dict	<pdf:Point pdf:Mark="name"> <pdf:Props> .... </pdf:Props> for inline props dict </pdf:Point>
BMC - Marked content	<svg:g pdf:Mark="Para"> ... svg content ... </svg:g>
BDC - Marked content with dict	<svg:g pdf:Mark="Para" xml:id="id123"> <pdf:Props> .... </pdf:Props> for inline props dict ... svg content ... </svg:g>

6.4 Page Destinations

Named destinations are pairs consisting of a string name and a PDFC destination which identifies a page and possibly view parameters to be applied to a view of that page.

To facilitate page assembly and disassembly of documents, Mars stores named destinations that reference a particular page with the other information for that page. Thus, the named destinations for a page can easily travel with a page. The document assembly process does need to assure uniqueness among all named destination names that appear in a document.

The named destinations for a given page are stored in a package file associated with that page. This file needs be read only if a named destination it contains is to be used. A cache file lists all of the named destination names and the page on which they are defined. This allows the location of each named destination in the document to be determined without requiring reading all of the named destination files.

Example 6 Named Destination File Example (dests.xml)

```
<Destinations>
    <Dest Name="F2">
        <XYZ Left="" Top="" Zoom=""/>
    </Dest>
    <Dest Name="G2.997341">
        <XYZ Left="161" Top="118" Zoom=""/>
    </Dest>
</Destinations>
```

Example 7 Named Destination Cache Example

```
<Cache>...<Data>
    <Dest Name="F2" Page_ref=”/page/0/info.xml”/>
    <Dest Name="G2.997341" Page_ref=”/page/1/info.xml”/>
</Data></Cache>
```

6.5 Page Content Annotations

PDF defines a set of annotations that appear effectively on a plane above the page. These annotations provide additional graphic and interactive content in the document. In Mars, annotations are stored with the page in up to two separate files associated with the page.

For a more complete discussion of annotations, see “Annotations” on page 96.

Mars segregates the annotations into two groups. Content annotations are considered part of page content and are the following annotations:

- Link Annotation (hyperlinks)
- Movie Annotation (embedded multimedia movie using platform player)
- Screen Annotation (area where media clips may be played)
- Widget Annotation (interactive form fields)
- PrinterMark Annotation (registration target, color bar, cut mark, etc.)
- Trap Network Annotation (define trapping characteristics for a page)
- Watermark Annotation (background graphics)
- 3D Annotation (3D objects)

Content annotations are stored in a separate file in the page directory whose conventional name is “pg.can”. The content annotations file is referenced from the <Page> element in the page information file:

Example 8 Page Info file references to Annotations, Content

```
<Page>
   <Annotations src="/page/0/pg.can"/>
   <Contents src="/page/0/pg.svg"/>
...
</Page>
```

6.6 Page Markup Annotations

The second group of annotations is Markup Annotations and includes the following annotation types:

- Text
- FreeText
- Line
- Square
- Circle
- Polygon
- Polyline
- Highlight
- Underline
- Squiggly
- StrikeOut
- Caret
- Stamp
- Ink
- Popup
- FileAttachment
- Sound
- Redaction

Markup annotations are stored in a separate file in the document package. Unlike content annotations, the markup annotations file is not directly named by the document XML. Instead, the package relationship mechanism is used. There is an annotation relationship defined between a file with name N and a file with name N.ann. Markup annotations for a page whose SVG page contents file is typically named pg.svg would appear in pg.svg.ann.

When a document is opened, to answer the question of whether markup annotations are present on a page, the presence of the file whose root matches the page content file and whose suffix is ".ann" in the document package must be checked.

The schema for markup annotations is referenced in Appendix A, "Schema Guide.

7 Graphic Content

Graphic content is represented in SVG format, a W3C standard for representing text and graphics. See http://www.w3.org/Graphics/SVG/. To meet the needs of representing and rendering documents in a compact and efficient way, a profile of SVG is used that corresponds roughly to a subset of SVG Tiny 1.2 plus a few features from SVG 1.1 and 1.2. This subset is called the "Fast Static Subset" (SVG/FSS). In addition, to support the representation and display of graphic constructs supported by PDF documents, some private namespace extensions to SVG/FSS are defined.

Some infrequently used features of PDF are not supported (see "Conversion Between PDF and Mars" on page 138).

7.1 SVG Representation of Page Contents

Page contents characterizes all of the text and graphic content and supplementary structure of pages, annotation, or form fields. Basic graphic content is specified using a subset of SVG Tiny 1.2 that is extended to support additional capabilities defined by PDF. The basic SVG constructs are in the SVG namespace. Mars-defined extensions are in the PDF namespace.

Mars with SVG graphic content can be read and displayed in Adobe Reader and converted to PDFC graphic operators using Acrobat Professional. Other applications and tools can create or manipulate SVG page contents as well.

Some logical structure information is interleaved with the SVG. This information can be used in conjunction with structure information in other files in the Mars package for reflow, accessibility, content recovery, and other purposes.

7.2 Handling of Unsupported SVG Content

Ideally, only supported SVG content should appear in Mars documents. In the event that unsupported SVG content does appear, user agents should ignore that content and continue processing the legal SVG/FSS content. Optionally, a warning could be displayed informing the user that unsupported content was encountered and ignored.

7.3 SVG Graphic Resources

PDFC graphic content references zero or more named resources. These resources are fonts, color spaces, patterns, shaders, procedure sets, images, and graphic "subroutines" called form xobjects. PDFC resources and their names are defined in a resources dictionary for the page or annotation in which they appear.

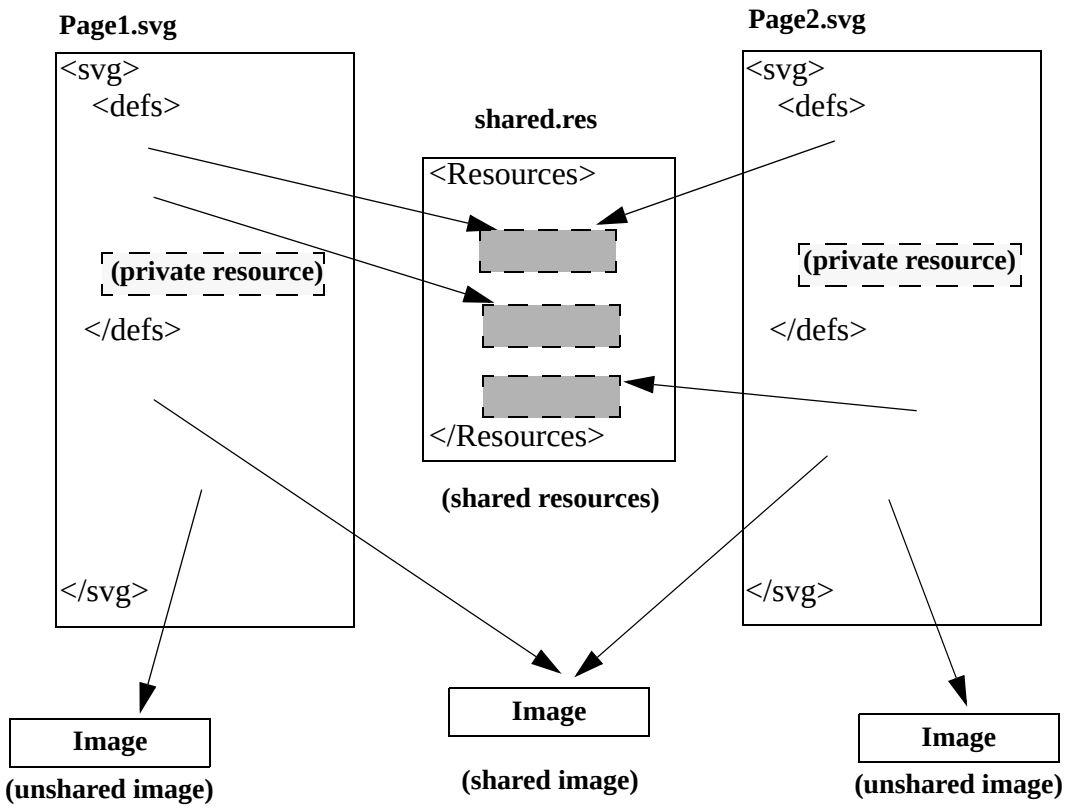
In SVG/FSS, some resources (e.g., images) are directly referenced by the SVG. Most other resources are declared in the <svg:defs> element of each page or annotation in which they are used. These definitions may be self-contained or may reference further information stored in a resource file elsewhere within the document package. Thus, multiple page or annotation SVG files can reference common resource definitions. Finally, some resources in PDFC (e.g., graphic states and procedure sets) are not required in Mars files and do not appear.

PDFC allows resource definitions to directly include other resource definitions. For example, a shader can directly include a function definition. In Mars, each resource definition must be separate and each resource must be named. All references from one resource to another are by name or URI reference.

Resource names in Mars need to follow XML ID attribute rules and be unique within each XML file in the document package. Thus, different pages (in different SVG files) can use the same ids for their own resources. In the case of resources shared between pages in a common resource file, the same rules apply. The resource ids must be unique in the shared resource file; ids in the shared re-

source file can conflict with ids on referencing pages because they are separate xml files.

FIGURE .1 *Sharing of resources from SVG/FSS content.*



When creating Mars files, consideration should be made to performance when placing resources in files.

1. Rendering of a page cannot really begin until all locally-defined resources are read, parsed, and processed. Consequently, pages with large numbers of resources that are used for only isolated parts of the page will benefit from declaring those resources externally (from the page).
2. If a separate resource file is used, all resources within it must be read and parsed when any reference is made to that resource file. Consequently, avoid placing resources in a file referenced by a page that are not used on that page.

3. Creating large numbers of small resource files will result in poor compression and larger file size as compression tends to work better on moderate sized files.

7.4 SVG Features Excluded from the Mars SVG/FSS Subset

The following table summarizes the major SVG Tiny 1.2 features that are excluded from SVG/FSS. A later section lists differences in more detail.

TABLE 8 Summary of Major SVG Features excluded from SVG/FSS

NAME	DESCRIPTION	REASON FOR EXCLUSION
CSS	All CSS-related markup (stylesheet PI, <svg:style> element, 'style' attribute) is not supported.	CSS processing is complex and time consuming.
Interactivity and Scripts	All markup related to interactivity and scripting (e.g., <svg:script>, event attributes, 'pointer-events' property, <svg:view> element) is not supported.	SVG in Mars is for static page display. Interactivity and scripting model are taken from PDF.
Filter Effects	All markup related to filter effects (e.g., <svg:filter> element and 'filter' property, 'color-interpolation-filters', flood-color', 'lighting-color') is not supported.	Performance issue and not supported in PDF.
Animation and Multimedia	All markup related to animation and multimedia is not supported.	PDF is primarily for static documents. Multimedia is taken from PDF.
tref, textPath	The <svg:tref> and <svg:textPath> elements are not supported.	Performance issue; not supported in PDF (alternate approach).

TABLE 8 Summary of Major SVG Features excluded from SVG/FSS

NAME	DESCRIPTION	REASON FOR EXCLUSION
Text Layout: Ligatures Line Layout Glyph Substitutions Text Alignment Kerning BIDI Groups Text Area	SVG properties related to spacing and layout are not supported. Text nodes which combine BIDI groups are not supported. Mars user agents must not perform ligature formation beyond those specified within the content within the content via the altGlyphs attribute on text and tspan elements. SVG font definitions must not define any glyphs with more than one Unicode character within the 'unicode' attribute. This guarantees the ability to map the glyph to a unique Unicode character. No glyph substitutions are to be performed.	PDF includes precision text layout and glyph selection.
Arbitrary order of definitions and use	All non-URI references for symbols, fonts, glyphs, gradients, patterns, clipping paths and masks must be local in the same file and must resolve to an element earlier in the file so that the file can be processed in a single pass.	Performance and single pass processing.

7.5 Mars Extensions to SVG/FSS

7.5.1 Summary

Several pdf namespace extensions to SVG are defined to enable full-fidelity representation of PDFC documents. The changes are summarized here and presented in more detail in a later section.

TABLE 9 Mars Extensions to SVG/FSS

NAME	DESCRIPTION (AND PARTIAL RELAXNG SCHEMA)
Top Level Declarations	Additional attributes must appear on the root <svg:svg> element that indicate usage of specific imaging features on the page. These are required so that tools can optimize processing of the document without having to fully analyze the page contents. See “Mars Rendering Information Extensions” on page 52. **pdf:UsesTransparency** **pdf:UsesSpotColors** **pdf:UsesOverPrint** **pdf:UsesKnockoutTransparencyGroups** **pdf:UsesIsolatedTransparencyGroupsNonRGBBlends** **pdf:UsesDefaultColorSpaces** **pdf:PageTransparencyGroupIsKnockout** **pdf:PageTransparencyGroupIsIsolated** **pdf:PageTransparencyGroupBlendColorspace**

TABLE 9 Mars Extensions to SVG/FSS

NAME	DESCRIPTION (AND PARTIAL RELAXNG SCHEMA)
Transparency	The following extensions enable expression of the full PDF 1.4 transparency model. These attributes can appear anywhere the %SVG.Opacity.attrib set can appear. attribute **pdf:BlendMode** {blend_mode}? blend_mode = [blend mode name or array of names; see schema] attribute **pdf:AlphaIsShape** {boolean}? The following attributes can appear only on an <svg:g> element and apply to transparent objects contained in the svg group. attribute **pdf:BlendingColorSpace** {color_space} color_space = [see schema and PDFR 1.6 Table 7.13 CS key and S7.2.3. This needs to be able to be represented as an attribute value. attribute **pdf:Isolated** {boolean}? attribute **pdf:KnockOut** {boolean}?
Soft Mask Extensions	The <pdf:softMask> element is a special variant of the SVG <mask> element. It contains a transparency group (a <g> element that has the special pdf transparency attributes defined, see above). The <pdf:softMask> element may have the following attributes: attribute **pdf:Subtype** ('Alpha' | 'Luminosity') attribute **pdf:BackdropColorSpace** {color_space}? attribute **pdf:BackdropColor** {array_of_number}? attribute **pdf:TransferFunction** {function}? BackdropColorSpace and BackdropColor only appear if Subtype is Luminosity. The array size of BackdropColor is the number of components in the BackdropColorSpace. For images used as soft masks, the <image> element can be enclosed in a regular SVG <mask> element. The <image> element may carry an additional optional attribute: attribute **pdf:Matte** {array_of_number}? pdf:Matte specifies a color value. The size of the array is the number of components in the color space of the 'host' image to which the soft mask is applied.

TABLE 9 Mars Extensions to SVG/FSS

NAME	DESCRIPTION (AND PARTIAL RELAXNG SCHEMA)
High-end Print Support	Various “pdf:” elements and attributes for high-end print workflow are supported [See PDFR 1.6 Table 4.8]. These attributes can appear anywhere the %SVG.Opacity.attrib set can appear. attribute **pdf:StrokeAdjustment** {boolean}? attribute **pdf:Overprint** {boolean}? attribute **pdf:OverprintPaint** {boolean}? attribute **pdf:OverprintMode** {integer}? attribute **pdf:UndercoverRemoval** {function_name}? function_name = [‘Default’ or a function reference in the <svg:defs> element or referenced file and id; see schema, see PDFR 1.5 T4.8 UCR, UCR2 keys] attribute **pdf:FlatnessTolerance** {number}? attribute **pdf:SmoothnessTolerance** {number}? attribute **pdf:BlackGeneration** { function_name }? attribute **pdf:HalfTone** { resource_name }? resource_name = [‘Default’ or name of a halftone resource defined in the svg:defs element or referenced file and id.] attribute **pdf:TransferFunction** { function_names }? function_names = [‘Identity’ or 1 or 4 function names which reference function resources in the svg:defs element or elsewhere.]
Image Extensions	<svg:image pdf:Interpolate=”true” ...> attribute is required on the <svg:image> element to express the /Interpolate property on a PDF image. attribute **pdf:Interpolate** {boolean}? attribute pdf:Decode {color_space_map}
Text Extensions	See Section 7.5.7 “Mars Text Extensions” on page 59. attribute **pdf:Transform** {...} on <svg:tspan> element <svg:text> or <svg:tspan> with attribute altGlyphs
ColorSpace Extensions	See “Color Spaces” on page 84
Smooth Shading Extensions	See “PDF Smooth Shading Extensions” on page 53

TABLE 9 Mars Extensions to SVG/FSS

NAME	DESCRIPTION (AND PARTIAL RELAXNG SCHEMA)
Function Extensions	See "PDF Function Extensions" on page 56
Blending Mode Marker	SVG/FSS page content that uses blending modes other than Normal must specify enable-background="new" on the outermost <svg> element; otherwise, all blending modes will be converted to Normal. (This is necessary for SVG compatibility. The enable-background="new" setting tells the SVG user agent to be prepared for the possibility of upcoming transparency groups.)
Font Descriptor for non-embedded font	An additional pdf-namespace element is defined that is part of the <svg:font-face> element. This additional element, <pdf:font-information> has an xlink:href attribute referring to a font descriptor file in the document package. Information in this file is used when a font substitution is required to better select a substitution font and process it consistent with the original font. See "Font Representation in Mars" on page 77.

7.5.2 Mars Rendering Information Extensions

Several extension attributes can appear on the root <svg> element of a page or annotation appearance. These attributes make it possible to accelerate the rendering of the page by enabling the proper configuration of rendering code prior to reading and parsing the <svg> content for the page. The attributes are described below.

If one of these attributes is missing or incorrectly set, the page must still render correctly, but may take significantly longer to appear.

TABLE 10 Rendering Information Attributes on the Root <svg> Element

pdf:UsesTransparency	Is there transparency on the page?
pdf:UsesSpotColors	Are spot colors used on the page and optionally in annotations on the page? [Implementation note: PDContentGetSpots today walks through /Resource dictionaries looking at color spaces.]

TABLE 10 Rendering Information Attributes on the Root <svg> Element

pdf:UsesOverPrint	Is overprint used on the page and optionally in annotations? [Implementation note: PDContentHasOverprint looks at the extended gstates and color spaces. Overprint is detected if a) there are separation or DeviceN color spaces and /OP or /op is true in some extended gstate, or b) OPM is seen with a value of 1 and /OP or /op is true.]
pdf:UsesKnockoutTransparency-Groups	PDContentIsGPUable looks for knockout transparency groups
pdf:UsesIsolatedTransparency-GroupsNonRGBBlends	or isolated ones with non-RGB blending spaces. (If no blending space is specified a CMYK one is assumed.)
pdf:UsesDefaultColorSpaces	Does the page have default color spaces? [Additional research and specification needed here.]
pdfPpageTransparencyGroupIs-Knockout **pdf:PageTransparencyGroupIsIsolated** **pdf:PageTransparencyGroup-BlendColorspace**	If there is a page level transparency group (this would be a /Group entry in the page dictionary with subtype /Transparency), these attributes indicate if it is a knockout group, if it is isolated and it's blending colorspace.

7.5.3 PDF Smooth Shading Extensions

PDF smooth shading extends SVG's gradient capabilities to draw more complex shading of objects. A shading object defines the properties of the gradient fill. This is a complex "paint" that determines the type of color transition the shading pattern produces when painted across an area.

Named shading objects must appear in the <svg:defs> element or in a separate resource file. References to locally-defined shading objects must appear following their declaration in the SVG file.

There are 7 PDF shading types represented by 7 pdf namespace elements. The schema for the SVG/FSS shading extensions follows:

```
shading_dictionary =
    element FunctionShader { type_1_shading_dictionary }
shading_dictionary |=
    element AxialShader { type_2_shading_dictionary }
shading_dictionary |=
    element RadialShader { type_3_shading_dictionary }
shading_dictionary |=
    element FreeFormShader { type_4_shading_dictionary }
shading_dictionary |=
    element LatticeShader { type_5_shading_dictionary }
shading_dictionary |=
    element CoonsShader { type_6_shading_dictionary }
shading_dictionary |=
    element TensorShader { type_7_shading_dictionary }
common_shading_dictionary =
    attribute ColorSpace { color_space_id }
  & attribute Background { general_color_array }?
  & attribute BBox { rectangle_blank_sep }?
  & attribute AntiAlias { boolean }?
general_color_array =      list { number }
type_1_shading_dictionary =
    element Domain { rectangular_domain_array }?
  & attribute Matrix { coordinate_map_array }?
  & element ColorFunctions { function_or_array_of_function }
  & common_shading_dictionary
type_1_shading_dictionary |= attribute xlink:href { uri }

type_2_shading_dictionary =
    element Axis { axis_coord_array }
  & element Domain { shading_domain_2 }?
  & element ColorFunctions { function_or_array_of_function }
  & element Extend { shading_extend_2 }?
  & common_shading_dictionary
type_2_shading_dictionary |= attribute xlink:href { uri }

type_3_shading_dictionary =
    element Coords { shading_radial_coords_array }
  & element Domain { shading_parametric_limits_array }?
  & element ColorFunctions { function_or_array_of_function }
  & element Extend { shading_extend_2 }?
  & common_shading_dictionary
type_3_shading_dictionary |= attribute xlink:href { uri }

type_4_shading_dictionary =
    attribute BitsPerCoordinate { integer }
  & attribute BitsPerComponent { integer }
  & attribute BitsPerFlag { integer }
  & attribute Decode { shading_decode_array }
  & element ColorFunctions { function_or_array_of_function }?
  & common_shading_dictionary
```

```
   & common_stream_dictionary
type_4_shading_dictionary |= attribute xlink:href { uri }

type_5_shading_dictionary =
     attribute BitsPerCoordinate { integer }
   & attribute BitsPerComponent { integer }
   & attribute VerticesPerRow { integer }
   & attribute Decode { shading_decode_array }
   & element ColorFunctions { function_or_array_of_function }?
   & common_shading_dictionary
   & common_stream_dictionary
type_5_shading_dictionary |= attribute xlink:href { uri }

type_6_shading_dictionary =
     attribute BitsPerCoordinate { integer }
   & attribute BitsPerComponent { integer }
   & attribute BitsPerFlag { integer }
   & attribute Decode { shading_decode_array }
   & element ColorFunctions { function_or_array_of_function }?
   & common_shading_dictionary
   & common_stream_dictionary
type_6_shading_dictionary |= attribute xlink:href { uri }

type_7_shading_dictionary =
     attribute BitsPerCoordinate { integer }
   & attribute BitsPerComponent { integer }
   & attribute BitsPerFlag { integer }
   & attribute Decode { shading_decode_array }
   & element ColorFunctions { function_or_array_of_function }?
   & common_shading_dictionary
   & common_stream_dictionary
type_7_shading_dictionary |= attribute xlink:href { uri }

shading_decode_array =       list { number }
shading_parametric_limits_array =
       attribute t0 { number }
   & attribute t1 { number }
shading_radial_coords_array =
       attribute x0 { number }
   & attribute y0 { number }
   & attribute r0 { number }
   & attribute x1 { number }
   & attribute y1 { number }
   & attribute r1 { number }
shading_extend_2 =
       attribute BeyondStart { boolean }
   & attribute BeyondEnd { boolean }
shading_domain_2 =
     attribute t0 { number }
   & attribute t1 { number }
axis_coord_array =
       attribute x0 { number }
```

```
    & attribute y0 { number }
    & attribute x1 { number }
    & attribute y1 { number }
  rectangular_domain_array =
      attribute Xmin { number }
    & attribute Xmax { number }
    & attribute Ymin { number }
    & attribute Ymax { number }
```

Examples of Shaders

Type 2 (axial) shading:

```
<pdf:AxialShader id="sh-52734" ColorSpace="DeviceCMYK" AntiAlias="false"
                 Matrix="1 0 0 1 0 0">
    <Axis x0="0" y0="0" x1="1" y1="0"/>
    <Domain t0="0" t1="1"/>
    <Extend BeyondStart="true" BeyondEnd="true"/>
    <pdf:ColorFunctions>
        <pdf:SampledFunction Domain="0 1" Range="0 1 0 1 0 1 0 1" Size="256"
                BitsPerSample="8" Order="linear" Encode="0 255"
                Decode="0 1 0 1 0 1 0 1" src="/res/func-60.f0"/>
    </pdf:ColorFunctions>
</pdf:AxialShader>
```

Type 3 (radial) shading:

```
<pdf:RadialShader Coords="25 25 0 50 50 50" id="sh-52073"
        ColorSpace="DeviceCMYK" AntiAlias="false" mtx="1 0 0 1 0 0">
    <Domain t0="0" t1="1"/>
    <Extend BeyondStart="false" BeyondEnd="false"/>
    <pdf:ColorFunctions>
        <pdf:SampledFunction Domain="0 1" Range="0 1 0 1 0 1" Size="255"
            BitsPerSample="8" Order="linear" Encode="0 254" Decode="0 1 0 1 0 1">
            <pdf:File Name="/res/func-0.f0"/>
        </pdf:SampledFunction>
    </pdf:ColorFunctions>
</pdf:RadialShader>
```

7.5.4 PDF Function Extensions

PDF Functions are represented by 4 pdf-namespace elements. pdf:Function elements can contain other functions, as allowed.

Type 1 (sampled) function example

```
<pdf:SampledFunction Domain="0 1" Range="0 1 0 1 0 1" Size="255" BitsPerSample="8"
            Order="linear" Encode="0 254" Decode="0 1 0 1 0 1">
   <pdf:File Name="/res/func-0.f0"/>
</pdf:SampledFunction>
```

Type 2 (exponential interpolation) example

```
<InterpolatedFunction C0="0.448 0.174 0.51 0.303" C1="0.184 0 0.016 0" Exponent="0.5"
                      Domain="0 1"/>
```

Type 3 (stitching) example

```
<pdf:StitchingFunction Domain="x1 x2 ... x(2 * m)" Range="x1 x2 ... x(2 * n)"
Bounds="x1 x2 ... x(k - 1)" Encode="x1 x2 ... x(2 * k)">
   <pdf:StitchingFunctions>
      ... (k) functions ...
   </pdf:StitchingFunctions>
</pdf:StitchingFunction>
```

Type 4 (PostScript calculator)

```
<pdf:PostscriptFunction Domain="0 1" Range="0 1 0 1 0 1">
   <pdf:File Name="/res/func-1.ps"/>
</pdf:PostscriptFunction>
```

```
function_dictionary =
    element SampledFunction { sampled_function_dictionary }
function_dictionary |=
    element InterpolatedFunction { interpolated_function_dictionary }
function_dictionary |=
    element StitchingFunction { stitching_function_dictionary }
function_dictionary |=
    element PostscriptFunction { postscript_function_dictionary }

sampled_function_dictionary =
      attribute Domain { array_of_number }
   & attribute Range { array_of_number }
   & attribute Size { array_of_integer }
   & attribute BitsPerSample { integer }
   & attribute Order { number }?
   & attribute Encode { array_of_number }?
   & attribute Decode { array_of_number }?
   & common_stream_dictionary
sampled_function_dictionary |= attribute xlink:href { uri }

interpolated_function_dictionary =
      attribute Domain { array_of_number }
   & attribute Range { array_of_number }
```

```
   & attribute C0 { array_of_number }?
   & attribute C1 { array_of_number }?
   & attribute Exponent { number }
interpolated_function_dictionary |= attribute xlink:href { uri }

stitching_function_dictionary =
     attribute Domain { array_of_number }
   & attribute Range { array_of_number }
   & element StitchingFunctions { array_of_function_dictionary }
   & attribute Bounds { array_of_number }
   & attribute Encode { array_of_number }
stitching_function_dictionary |= attribute xlink:href { uri }

postscript_function_dictionary =
       attribute Domain { array_of_number }
   & attribute Range { array_of_number }
   & common_stream_dictionary_no_filters
postscript_function_dictionary |= attribute xlink:href { uri }

array_of_function_dictionary =
     element Function { function_dictionary }*
```

7.5.5 PDF HalfTone Extensions

Halftones are references from a pdf:halfTone attribute whose value references a halftone resource. The resource may be in a child element of the <svg:defs> element or may be a resource in a separate resource file. The schema for halftones follows.

FIGURE .6 *Halftone Resource Schema*

```
halftone_resource = element Halftone { attribute id {text}? & attribute Name { text }?
                                  & halftone_dictionary }
halftone_dictionary =
     attribute HalftoneType { integer }
   & attribute HalftoneName { pdf_byte_string }?   & attribute HalftoneName_enc { token }?
   & attribute Frequency { number }
   & attribute Angle { number }
   & element SpotFunction { halftone_dictionary_spot_function }
   & attribute AccurateScreens { boolean }?
   & element TransferFunction { halftone_dictionary_transfer_function }?
   & attribute Width { integer }?
   & attribute Height { integer }?
   & attribute Xsquare { integer }?
```

```
    & attribute Ysquare { integer }?
    & attribute Width2 { integer }?
    & attribute Height2 { integer }?
    & element ColorantDefault { halftone_dictionary_colorant }
    & element Colorant { attribute Name { text }, halftone_dictionary_colorant }*
halftone_dictionary_spot_function = function_dictionary
halftone_dictionary_spot_function |= name

halftone_dictionary_transfer_function = function_dictionary
halftone_dictionary_transfer_function |= name

halftone_dictionary_colorant = halftone_dictionary
halftone_dictionary_colorant |= common_stream_dictionary
```

7.5.7 Mars Text Extensions

Transform on tspan

To match expressivity of PDF, a transform attribute is allowed on svg:tspan elements. The attribute value is the same as in other uses of the svg:transform attribute covered in SVG spec section 7.6.

attribute **pdf:Transform** {...} on <svg:tspan>

Alternate Glyphs

There are two main circumstances under which glyph IDs are used:

1. Using glyphs that cannot be directly addressed with Unicode. Such glyphs ordinarily are accessed only by glyph substitutions in the OpenType font. Since Mars text is final-form (to match PDF), such substitutions must be performed while generating the Mars text; they are not allowed when presenting it.
2. Translating PDF text whose Unicode equivalents cannot be determined. This arises when a PDF font uses a custom encoding and has no ToUnicode CMap.

SVG already include a mechanism for specifying alternate glyphs for Unicode text. Such glyph specification is more common in PDF and requires a textually shorter expression.

On <svg:text> and <svg:tspan> elements:

attribute pdf:altGlyphs { text }

The value of altGlyphs is a space-separated list of strings (uri refs or glyph-ids) with an optional parenthetical pair of numbers: (#chars[, #glyphs]) that can precede string. For example, "(3,2) 125 126" says to consume 3 characters from the Unicode string and replace with the next two glyphs (glyph ids 125 and 126).

- If a given glyph is not preceded by a (n,m) expression that includes it, the default is (1,1).
- If only "#chars" is provided, the default for #glyphs is 1.
- If #chars=0, no Unicode characters are consumed, but the following glyph(s) is/are rendered.
- If #chars>0 and #glyphs=0, then the next #chars Unicode characters are rendered without glyph substitution.
- Any situations where the glyph attribute is incorrectly formed (e.g., you run out of glyphs or characters) is treated as an unsupported attribute and the viewer must render the Unicode characters as if the glyph attribute were not present.
- Be aware that Unicode normalization could damage text and tspan element content by changing the number of characters and making it inconsistent with any altGlyphs attributes.

Example 9 SVG Text with altGlyphs attribute and no Unicode characters

```
<text transform="…" font-size="12" font-family="F1" fill="…" fill-rule="evenodd">
    <tspan x="0 12.0769 16.4434 27.5957" pdf:altGlyphs="4 14 16 12"/>
</text>
```

Example 10 SVG Text with altGlyphs attribute with Unicode characters

```
<text transform="…" font-size="12" font-family="F1" fill="…" fill-rule="evenodd">
    <tspan x="0 0 0 0 12 16 27 34" pdf:altGlyphs="(3,4) 4 14 16 12 13 14 15 78">abcdefg
    </tspan>


</text>
```

In this example, a, b, and c are replaced with glyphs 4, 14, 16, and 12. Then d, e, f, and g are rendered using glyphs 13, 14, 15, and 78, respectively.

Note: *The altGlyphs extension enables a more compact representation than the standard <altGlyph> mechanism in SVG. This is not only because it allows multiple glyphs to be specified at once but also because it is represented as an at-*

tribute rather than an element. The decision to define altGlyphs as an attribute was made so that if Mars text is fed into a standard SVG processor, the altGlyphs will be ignored rather than causing an error.

7.6 Linkage from Page Content to Page Components

SVG page content references other objects in the document package. References outside the document package are not allowed.

TABLE 11 References From SVG To Other Document Package Objects

REFERENCE FROM	REFERENCE TARGET	NOTES
<color-profile xlink:href=”target”	ICC Profile File	
<text font-family=”F1”>	<font-face font-family=”F1”	Internal reference of short name. The name is looked up in the <svg:defs> element. There must be a matching <font-face-uri> element in the defs element of the SVG file.
<font-face font-family="F2"> <font-face-src> <font-face-uri xlink:href="../font/14.otf"/> <font-face-name name="Myriad Pro"/> </font-face-src> </font-face>	file /font/14.otf within package	Direct reference to the OpenType font.
<image xlink:href="myimage.png">	Image file within package. Can be .jpeg, .png, .jp2, .jbig2	Needs to map in and out of image resource dictionary indirection going between PDFC and Mars.
<image color-profile=”cs-2”	<pdf:pdfColorProfile name="cs-2"	Internal reference to color profile in <svg:defs> element.
<path fill="url(#sh-247)" d="M-0.318,0.0343h1 [...] > <path fill="url(/shaders/common.res#sh-247)" d="M-0.318,0.0343h1 [...] >	<defs><FunctionShader ID=”sh-247” ... in file /shaders/common.res: <FunctionShader ID=”sh-247” ...	Reference to internally-defined pdf shader within the <svg:defs> element. Reference to shader defined in a separate resource file.

7.7 SVG Tiny 1.2

This section lists differences between SVG/FSS and the SVG Tiny 1.2 Specification (21 July 2006) and indicates restrictions or extensions to the SVG language. These restrictions or extensions are present so that the full range of PDF document content can be represented and displayed efficiently.

If a section is not listed, there are no variances of significance from the SVG specification. Note that only the main instances of descriptions of SVG features are called out. Many times, subsequent sections make short mentions of earlier features; these are not listed as variances.

TABLE 12 SVG/FSS and SVG Tiny 1.2

SVG TINY 1.2 SPEC SECTION	FSS
1.1 About SVG	Interaction, dynamic, scripting, and animation are not included.
2.1.6 Scriptable	Scripting is not supported.
2.2.4 Animation	Animation is not supported.
2.3 Options for using SVG in Web pages	Since use in Mars is not in the context of web pages, this section is not relevant.
4.1 Basic Data Types	Numbers are not limited to 4 decimal places; range is not limited to +/-32767 <scientific-number> is not supported.
5.1.2 The 'svg' element	SVG content that conforms to the Fast Static Subset that uses blending modes other than Normal must specify enable-background=”new” on the outermost <svg> element; otherwise, all blending modes will be converted to Normal. (This is necessary for SVG compatibility. The enable-background=”new” setting tells the SVG user agent to be prepared for the possibility of upcoming transparency groups.) Nested <svg> elements are not supported. Unsupported attributes: snapshotTime, playbackOrder, timelineBegin, contentScriptType, focusable, navigation attributes, preserveAspectRatio, zoomAndPan, viewBox The PDF media box and crop box subsume the function of viewBox.

TABLE 12 SVG/FSS and SVG Tiny 1.2

SVG TINY 1.2 SPEC SECTION	FSS
5.2 Grouping: the 'g' element	The <g> element may contain pdf:Mark attribute. See “Marked Element and Logical Structure Representation” on page 39.
5.3 The 'defs' element	Definitions must be inside the <defs> element and definitions must appear before they are referenced.
5.4 The 'discard' element	<discard> is not supported.
5.5 The 'desc' and 'title' elements	<desc> and <title> are not supported
5.7 The 'image' element	Image Interpolation: <svg:image pdf:Interpolate=”true” ...> attribute is required on the <svg:image> element to express the /Interpolate property on a PDF image. attribute pdf:Interpolate {boolean}? Supported image file types. See Section 11 “Images” on page 82.
5.8 Conditional processing	<switch> and its attributes are not supported.
5.9 External Resources	External resources (external ResourcesRequired attributes) are not supported.
5.9.3 The 'prefetch' element	<prefetch> is not supported.
6 Styling	All CSS-related markup (stylesheet PI, <svg:style> element, ‘style’ attribute) is not supported. CSS Units: CSS units or percentages on length values is not supported.
7.14 Geographic Coordinate Systems	Coordinate system metadata is not supported.
7.15 The svg:transform attribute	Coordinate system metadata is not supported.
10.2 Characters and their corresponding glyphs	Mars Alt glyph extension
10.5 The 'tspan' element	Extension: attribute pdf:transform {...} on <svg:tspan>
10.6 Text layout	The PDF model is a precision text layout model. Consequently, all svg features related to automatic text layout are not supported. This includes most of the features in this section.
10.8 Alignment properties	‘text-anchor’ is not supported (all content must conform to text-anchor=”start”)

TABLE 12 SVG/FSS and SVG Tiny 1.2

SVG TINY 1.2 SPEC SECTION	FSS
10.9 Font selection properties	font-weight, font-style, font-variant, font-size, font-stretch are supported. Only a single font-family name is allowed in a font-family property value. If a list is provided, then the content is not conforming and the processor must use only the first value in the list. font-size-adjust and font are not supported.
10.11 Text in an area	textArea, tbreak elements and line-increment, text-align, display-align is not supported.
10.12 Editable Text Fields	Editable text fields are not supported.
11.2 Specifying paint	Paint servers uri's can refer to pdf shaders and pattern definitions.
11.5 Non-Scaling Stroke	vector-effect is not supported.
11.7 The 'viewport-fill' Property	viewport-fill is not supported.
11.8 The 'viewport-fill-opacity' Property	viewport-fill-opacity is not supported.
11.9 Controlling visibility and rendering	Visibility control and the 'display' property are not supported.
11.12 Object and group opacity	Several extensions can appear the same places that the opacity attribute can appear. These extensions are: attribute pdf:BlendMode {blend_mode}? attribute pdf:blendingColorSpace {color_space} attribute pdf:alphaIsShape {boolean}? attribute pdf:isolated {boolean}? attribute pdf:knockOut {boolean}?

TABLE 12 SVG/FSS and SVG Tiny 1.2

SVG TINY 1.2 SPEC SECTION	FSS
11.13.1 Syntax for color values	Float functional percentage rgb() is not supported. SVG 1.2 style colors are partially supported and include some extensions: Colors are represented in fill and stroke attributes as follows: -- Colors that have n colorants and are part of a color space with an ICC profile have the format: rgb(r, g, b) icc-color(<colorspace-name>, n1, n2, n3 ... nn) The first r,g,b triplet is a device-color approximation of the ICC color. <colorspace-name> refers to a color-profile element that's defined in the document's 'defs' section. -- Colors that are part of a color space without an ICC profile have the format: rgb(r, g, b) device-color(<colorspace-name>, n1, n2, n3 ... nn) <colorspace-name> refers to a color space definition element (see later) that's defined in the defs section or it can be one of two special names: "DeviceCMYK" or "DeviceGray".
11.14.1 System Paint Servers	System Paint Servers are not supported.
11.14.3 The SVG 'color' property	Not supported. Content must specify the color directly via the ‘fill’ and ‘stroke’ properties.
11.16 Gradients	In addition to the facilities defined in SVG Tiny 1.1, Mars includes Patterns from SVG 1.1 and several extensions to support PDF features around shading and gradients.
12 Multimedia	Multimedia features are not supported. audio, video, animation elements;
13 Interactivity	Interactivity is not supported. events, pointer-events, focusable, nav-next, nav-etc.focus-highlight attributes.
14.1.4 Reference Restrictions	References outside the document package are never allowed. See sections covering individual attributes for details of other restrictions.
14.1.5 IRI reference attributes	xlink attributes other than href are ignored.
14.1.6 Externally referenced documents	

TABLE 12 SVG/FSS and SVG Tiny 1.2

SVG TINY 1.2 SPEC SECTION	FSS
14.2 Links out of SVG content: the 'a' element	The a element is not supported. Instead, use link annotations to represent external links.
14.3 Linking into SVG content: IRI fragments and SVG views	Addressing SVG content with fragment identifiers is not supported.
15 Scripting	Scripting is not supported. Features not supported include the script element, XML Events, handler element, listener element, Event handling
16 Animation	Animation is not supported. Features not supported include SMIL, animate element and its attributes, animation-related attributes on other elements, attributes begin, dur, end, min, max, restat, repeatCount, repeatDur, fill, element discard animateMotion, animateColor, animateTransform, mpath
17.8.2 The 'font-face' element	This is standard usage within SVG with the interpretation of the svg:font-face-uri as referencing a file contained in the document package. Initial declaration of font in the defs part of the SVG: <svg:font-face font-family=”F1”> <svg:font-face-src> <svg:font-face-uri xlink:href=”...URL of font within package...”/> </svg:font-face-src> </svg:font-face> A reference to the font looks like: <svg:text font-family=”F1”>hello</svg:text>
18 Metadata	Metadata and the <metadata> element are not supported. Instead, XMP metadata should be associated with the page.
19.2.1 The 'foreignObject' element	the foreignObject element is not supported.

7.8 SVG 1.1

This section lists SVG 1.1 features, not part of SVG Tiny 1.2, that are include in SVG/FSS.

TABLE 13 SVG 1.1 Features Included in SVG/FSS

SVG 1.1 SPEC SECTION	
5.5 The 'symbol' element	Symbol Resolution: All non-URI references for symbols, fonts, glyphs, gradients, patterns, clipping paths and masks must be local and must resolve to an element earlier in the document tree. Define-before-use: <svg:use> elements can only refer to <svg:symbol> elements within the same file. In general, names must be defined textually before they are used so that the file can be processed in a single pass.
5.6 The 'use' element	Instantiate a template defined by <symbol>
12.3 Color profile descriptions	Defines managed color profiles. See also Section 12 “Color Spaces” on page 84. <color-profile> element and color-profile attribute on <image> elements are included but not CSS styling
13.3 Patterns	<pattern> element
14.3.5 Establishing a new clipping path	<clipPath> element creates a new clipping path.
14.4 Masking	The SVG <mask> element is supported. PDF soft masks are mapped to <pdf:softMask> elements.
20.7 The 'hkern' and 'vkern' elements	vkern is part of FSS in addition to hkern.

7.9 Features from SVG 1.2

This section lists SVG 1.1 features, not part of SVG Tiny 1.2, that are include in SVG/FSS. Section numbers reference the 27 October 2004 release of the specification.

TABLE 14 SVG 1.2 Features included in SVG/SFF

SVG 1.2 (DRAFT) SPEC SECTION	
11.5 Using device colors	Use of device-color to support additional PDF color spaces.

7.10 Elements and Attributes In SVG 1.1, Tiny 1.2, and FSS

Reasons for exclusion from FSS are indicated in square brackets.

TABLE 15 SVG 1.1, SVG Tiny 1.2, and SVG/FSS

ELEMENT / ATTRIBUTE	1.1	TINY 1.2	FSS
attr audio-level		12.5	No [multimedia]
attr color		11.14.3	No [CSS]
attr color-rendering		11.10.1	No (hint)
attr display		11.9	No, use PDF optional content instead.
attr display-align		10.11.6	No [layout]
attr fill		11.3	Yes
attr fill-opacity		11.3	Yes
attr fill-rule		11.3	Yes
attr font-family		10.9	Yes
attr font-size		10.9	Yes
attr font-style		10.9	Yes
attr font-variant		10.9	Yes
attr font-weight		10.9	Yes
attr image-rendering		11.10.4	No (hint)
attr line-increment		10.11.4	No [layout]
attr opacity		11.12	Yes
attr pointer-events		13.9	No [interaction]
attr shape-rendering		11.10.2	No (hint)
attr solid-color		11.14.2	No
attr solid-opacity		11.14.2	No
attr stop-color		11.16.3	Yes
attr stop-opacity		11.16.3	No
attr stroke		11.4	Yes
attr stroke-dasharray		11.4	Yes

TABLE 15 SVG 1.1, SVG Tiny 1.2, and SVG/FSS

ELEMENT / ATTRIBUTE	1.1	TINY 1.2	FSS
attr stroke-dashoffset		11.4	Yes
attr stroke-linecap		11.4	Yes
attr stroke-linejoin		11.4	Yes
attr stroke-mitrelimit		11.4	Yes
attr stroke-opacity		11.4	Yes
attr stroke-width		11.4	Yes
attr text-align		10.11.5	No [layout]
attr text-anchor		10.8.1	No [layout]
attr text-rendering		11.10.3	No (hint)
attr vector-effect		11.5	No
attr viewport-fill		11.7	No
attr viewport-fill-opacity		11.8	No
attr visibility		11.9	No
element a	17.1	14.2	No
element altGlyph	10.14		No, see Section 7.5.7
element altGlyphDef	10.14		No
element altGlyphItem	10.14		No
element animate	19.2.10	16.2.11	No [Animation]
element animateColor	19.2.13	16.2.15	No [Animation]
element animateMotion	19.2.12	16.2.13	No [Animation]
element animateTransform	19.2.14	16.2.16	No [Animation]
element animation		12.4	No [Animation]
element audio		12.2	No [Multimedia]
element circle	9.3	9.3	Yes
element clipPath	14.3.5		Yes, subset

TABLE 15 SVG 1.1, SVG Tiny 1.2, and SVG/FSS

ELEMENT / ATTRIBUTE	1.1	TINY 1.2	FSS
element color-profile	12.3.3		Yes
element cursor	16.8.3		No [Interaction]
element definition-src	20.8.3		No
element defs	5.3.3	5.3	Yes
element desc	5.4	5.5	No [ignore?]
element discard		5.4	No
element ellipse	9.4	9.4	Yes
element feBlend, feColorMatrix, ..., feTile, feTurbulence	15		No [Filters]
element filter	15.3		No
element font	20.3	17.3	Yes , subset
element font-face	20.8.3	17.8.2	Yes, subset
element font-face-format	20.8.3		No ??
element font-face-name	20.8.3		Yes
element font-face-src	20.8.3	17.8.3	Yes
element font-face-uri	20.8.3	17.8.4	Yes
element foreignObject	23.3	19.2.1	No
element g	5.2.2	5.2.2	Yes, extensions
element glyph	20.4	17.4	Yes, in font defs
element glyphRef	10.14		No [part of alt glyphs]
element handler		15.5	No [scripting]
element hkern	20.7	17.7	Yes, in font defs
element image	5.7	5.7	Yes, plus extensions
element line	9.5	9.5	Yes
element linearGradient	13.2.2	11.16.1	Yes
element listener		15.4	No [events]
element marker	11.6.2		No

TABLE 15 SVG 1.1, SVG Tiny 1.2, and SVG/FSS

ELEMENT / ATTRIBUTE	1.1	TINY 1.2	FSS
element mask	14.4		Yes, plus extensions
element metadata	21.2	18.2	No
element missing-glyph	20.5	17.5	Yes, in font defs
element mpath	19.2.12	16.2.14	No [animation]
element path	8.2	8.2	Yes
element pattern	13.3		Yes, plus extensions
element polygon	9.7	9.7	Yes
element polyline	9.6	9.6	Yes
element prefetch		5.9.3	No
element radialGradient	13.2.3	11.6.2	Yes
element rect	9.2	9.2	Yes
element script	18.2	15.2	No [scripting]
element set	19.2.11	16.2.12	No [animation]
element solidColor		11.14.2	No
element stop	13.2.4	11.16.3	Yes
element style	6.11		No [CSS]
element svg	5.1.2	5.1.2	Yes, subset, extensions
element switch	5.8.2	5.8.2	No
element symbol	5.5		Yes, define before used
element tbreak		10.11.3	No [layout]
element text	10.4	10.4	Yes, subset
element textArea		10.11.2	No [layout]
element textPath	10.13.2		No [text on path]
element title	5.4	5.5	No
element tref	10.6		No
element tspan	10.5	10.5	Yes, subset
element use	5.6	5.6	Yes

TABLE 15 SVG 1.1, SVG Tiny 1.2, and SVG/FSS

ELEMENT / ATTRIBUTE	1.1	TINY 1.2	FSS
element video		12.3	No [multimedia]
element view	17.2.3		No
element vkern	20.7		Yes, in fonts

8 Embedded Files

Embedded files represent supplementary files that travel with the document but are not part of displayed document contents. Embedded files can be, but need not be, associated with a file attachment annotation which appears on a document page.

There are three components concerned with embedded files:

- The file itself which is represented as a separate file in the document package. The file must appear in the /file directory of the package or a subdirectory thereof. The embedded file can have any name that is a legal name as defined by Zip. Note that the filename within the package is used only to reference the file from the backbone or annotation. A separate attribute within the XML (Name and/or UName) holds the file name as it will be displayed for the user.
- In the document backbone, an element <EmbeddedFile> appears which references another file in the document package. This second file contains information about each embedded file. The root element of this file is <EmbeddedFiles> and it contains a child <EmbeddedFile> element for each embedded file. This element is a file_spec_dictionary which contains the file name and optional information such as a UI description of the file, mime type, creation date, checksum, and last modification date. See “Schema Guide” on page 136 for complete details.
- Optionally, a file attachment annotation which contains the same information as the <EmbeddedFile> element as well as the information needed to display the annotation on its page. The annotation information is stored in the .ann file

associated with the page on which it appears. See “Annotations” on page 96. An Embedded file annotation is a markup annotation.

Example 11 Backbone markup associated with an embedded file

```
in the backbone.xml file:
...
<EmbeddedFiles src="/file/embedded_files.xml"/>

in /file/embedded_files.xml:

<?xml version="1.0" encoding="UTF-8"?>
<EmbeddedFiles xmlns="http://ns.adobe.com/pdf/2006">
    <EmbeddedFile Key="Untitled Object"
                  UName="arch_pic.bmp"
                  Name="arch_pic.bmp">
        <FileData src="/file/ef_4209265555-3" FileType="image/bmp">
            <Params Creation="D:20060706161059-07'00'"
                    ModificationDate="D:20060706161059-07'00'" Size="506934"
                    Checksum="48uBet+wuxfygp5JKshUcQ=="/>
        </FileData>
        <Desc>This is a sample image that can be projected with the EXR-55</Desc>
    </EmbeddedFile>
</EmbeddedFiles>
```

Note: *The Key attribute on the EmbeddedFile element can be used to identify the file and often is used in server processing of files. It does not typically appear in the user interface.*

9 Forms

Adobe PDF Forms as implemented in Acrobat 5 and before (and supported in Acrobat 6 and beyond) include a PDF description of forms fields and their behavior. This description in Mars appears directly in the Mars backbone.

Example 12 Sample AcroForm Markup

```
<Acroform>
    <Fields>
        <Button Widget_ref="/page/0/pg.can#0" UIName="Press this button right now"
                Name="Button1"  Flags="ReadOnly PushButton">
        </Button>
```

```
        <Button Widget_ref="/page/0/pg.can#1" UIName="tooltip" Name="Check Box2">
        </Button>
        <Choice Widget_ref="/page/0/pg.can#3" UIName="tooltip for combo box"
        Name="Combo Box3"  Flags="Required Combo Edit Sort CommitOnSelChange">
            <OnCursorEnter>...</OnCursorEnter>
            <BeforeFormat>...</BeforeFormat>
        </Choice>
        <Text Widget_ref="/page/0/pg.can#4" UIName="tt text" Justification="Centered"
            Name="Text6"  Flags="Required RadiosInUnison">
            <DefaultValue>Here is a default</DefaultValue>
        </Text>
        <Signature Widget_ref="/page/4/pg.can#2" UIName="sign here"
                    Name="Signature7"/>
    </Fields>
</AcroForm>
```

In Mars, the default appearances for form fields is represented in SVG. These SVG fragments can appear inline in the form field widget entry or in separate package level files (which are directly referenced from the appearance element).

Note: *It is permissible to omit default appearances in Mars as there are no back-rev viewers that can make use of them. It is fine to let the viewer create the appearances of the fields. Be aware that some annotations have no default appearances (e.g., stamps with no icon) and other appearances may vary between implementations if the appearance is not explicitly specified.*

9.1 XFA/XML Form Objects

There are several objects that comprise an XML Form. The current format for XFA form information in a PDFC file is:

- The XFA key in the Acroform dictionary contains an array of pairs of string and stream entries where each string identifies the stream content. The string usually matches the tag of the XML in the corresponding stream. See PDFR 1.6 section 8.6.7. The first and last array entry are special and represent the opening <xdp:xdp> tag and closing </xdp:xdp> tag, respectively.
- The streams which contain the XML for the form template, datasets, configuration items, and so on.

The string names can be arbitrary except for:

- The XFA template must be identified by the string “template”
- The XFA datasets must be identified by the string “datasets”
- The XFA configuration must be identified by the string “config”

In Mars, each of the streams which comprise the form is represented in a separate file. By convention, the files are placed in the /form folder. The first and last entry which contain only the opening and closing xdp tag are included in the Mars file. Note that these files may not be complete XML documents. The XML in the backbone explicitly references each file.

Example 13 Mars Backbone entry describing an XML form

```
<XFA>
    <FormPart src=”/form/template.xml” Name=”template”/>
    <FormPart src=”/form/datasets.xml” Name=”datasets”/>
    <FormPart src=”/form/config.xci” Name=”config”/>
    <FormPart src=”/form/other_stuff.xml” Name=”plastic”/>
</XFA>
```

An alternative representation in PDFC is to package all of the XFA components in XDP format and place the result in a single stream in the XFA key of the Acroform dictionary. This alternative format is not supported in Mars.

The relevant XFA specs are:

XML Data Package (XDP)	http://partners.adobe.com/public/developer/xml/index_arch.html
XML Form Template (XFT)	http://partners.adobe.com/public/developer/xml/index_arch.html

9.2 AcroForms

Acroforms consist of the information in the AcroForm element and the individual form field widgets which appear as content annotations on the pages where they are displayed.

In Mars, unlike PDFC, the form field values are represented in a separate file rather than simply being entries in each field dictionary. The file follows the

XFDF schema for form fields. It does not contain annotations. The file name is "/form/form_data.xfdf".

For XFDF, see:

http://partners.adobe.com/public/developer/xml/index_arch.html

10 Fonts

Mars supports embedded and non embedded OpenType fonts and embedded SVG fonts. Each font used in the document should have either a font descriptor file, or the font file itself. The font descriptors are represented in XML and can be shared among pages in a Mars document. The fonts themselves are represented in OpenType or SVG format and can also be shared among pages and other graphic objects in a document.

If neither the font descriptor nor embedded font is present in the Mars document, then the actual font used is implementation defined and may not correspond very closely with the author's intent.

10.1 Font Support in Mars

10.1.1 OpenType Fonts

Mars supports embedded and non-embedded OpenType fonts and reduced OpenType fonts (where only a subset of the tables listed as "required" in the OpenType specification are present. See the OpenType specification at http://partners.adobe.com/public/developer/opentype/index_spec.html.

TABLE 16 Required OpenType Font Tables

TABLE	TABLE
glyf (TrueType-based fonts)	cvt (TrueType-based fonts)
CFF (CFF-based fonts)	prep (TrueType-based fonts)
cmap (TrueType-based fonts)	hmtx (TrueType-based fonts)
head (TrueType-based fonts)	fpgm (TrueType-based fonts)

TABLE 16 Required OpenType Font Tables

TABLE	TABLE
hhea (TrueType-based fonts)	maxp (TrueType-based fonts)
loca (TrueType-based fonts)	

OpenType fonts can be embedded in the document package. When this is done, the embedded font is for the exclusive use of the document and is obfuscated. Such fonts cannot be removed from the package and used in other documents or for other purposes. See “Font Protection” on page 81. Important note: processing applications should only embed fonts if the permission bit for embedded is set. See http://partners.adobe.com/public/developer/en/acrobat/sdk/FontPolicies.pdf.

Tables listed in Table 16 are required only to the extent that they were present in the original font.

10.1.2 SVG Fonts

Mars supports SVG fonts. SVG fonts can be embedded in the SVG page directly or referenced in a separate file. When embedded directly, the SVG font is not encrypted. When SVG fonts are referenced via a special format uri that refers to a separate package file, they are obfuscated.

10.1.3 Other Font Types

Other font types, including TrueType Collections and Type 3, are not supported.

10.2 Font Representation in Mars

10.2.1 Fonts and Font Descriptors

Fonts contain (encrypted) data that represents the actual font in OpenType format. This applies to fonts embedded in the document.

An optional *Font Descriptor* can also be included in the document and associated with a font. The Font Descriptor contains information about the font in gener-

al and not about individual glyphs. This information is used by viewers when displaying documents for which the original font could not be located and a substitution is required.

Within an SVG file, text to be displayed refers to a font by a short string name such as “F3” in the “font-family” attribute and by characteristics of the font such as its weight.

10.2.2 Storage of Fonts and Font Descriptors in the package

Fonts are stored in separate files in the document package. By convention, they should be placed in the /font directory or in the directory of the page on which they are referenced. Font descriptors are XML files which are likewise stored in separate package files. They also should be placed in the /font directory or in the directory of the page on which their associated font is referenced.

Keep in mind that the existence of a font or font descriptor in the package does not imply that it is used in any way. Only the uri references from other package objects establish a using relationship.

10.2.3 Font references from SVG

References to embedded font.	Initial declaration of font in the <svg:defs> element: <svg:font-face font-family=”F1”> <svg:font-face-src> <svg:font-face-uri xlink:href=”...URI of font within package...”/> </svg:font-face-src> </svg:font-face> Later reference to the font: <svg:text font-family=”F1”>hello</svg:text>
Reference to non-embedded font.	<svg:font-face font-family=”F2”> <font-face-name name="Times New Roman"/> <pdf:font-information xlink:href=”/font/f1234.fd”/> ... </svg:font-face> <svg:text font-family=”F2”>hello</svg:text>

10.3 Font Descriptors

Each font descriptor is a separate package level file. Font descriptor files may be in any non-reserved location in the package but by convention should be placed in either the /font directory or the directory for the page on which the font descriptor is used.

The font descriptor contains information about the font including widths and Unicode mappings that are useful if a font substitution should be required. This information allows the document to be rendered with layout similar to the original.

The font descriptor must maintain the original single byte encoding for Type 1, TrueType, and Type 3 fonts. For Type 0 fonts, the Character Collection Registry, Ordering and Supplement for CIDFonts, the glyph widths, and the FontDescriptor information must be maintained. This is to enable reliable selection and use of substitute fonts when the end-user views the document should the original font be unavailable.

Example 14 Sample Font Descriptor File Contents

```
<Font>
   <Type1 LastChar="255" BaseFont="Bembo" FirstChar="0">
      <FontDescriptor StemV="67" FontName="Bembo" Flags="Serif Nonsymbolic"
                  Descent="-233" Ascent="673" CapHeight="622" XHeight="396"
                  FontBBox="-167 -236 1074 963" >
      </FontDescriptor>
      <Widths>278 278 278 .....  677 594 677 594 </Widths>
      <Encoding>MacRomanEncoding</Encoding>
   </Type1>
</Font>
```

10.3.1 Simple Fonts that use the Latin character set

In simple fonts (other than Type 0 fonts), the Encoding entry in the Mars font descriptor maps from a single byte character code to a glyph name. The Encoding entry can be one of the predefined Encodings, be based on a pre-defined encoding with Differences, consist entirely of a Differences array, or be a built-in encoding. If the font only uses glyphnames that are in the Latin Character set as defined in Appendix D.1. of the PDF specification version 1.6, a multiple master substitution font can be created for the font. In order for the substitution font to be created the widths of each glyph are needed as well as the other properties

found in the Mars font descriptor. The name of the font, stored in the BaseFont attribute, is used to first try to locate a font on the user's system before creating the substitution font. The BaseFont name can be the PostScript name or the Windows(R) Logfont name with spaces removed. In order to find the font on the user's system or to create a substitution font the following additional information from the font descriptor is needed:

1. BaseFont name
2. Character widths
3. Stem Weights, Style, etc.
4. Mapping from character code to glyphnames (i.e. The Encoding vector (see Section 10.3.2 "Built-in Encodings" on page 80).

10.3.2 Built-in Encodings

If the Encoding is missing or if the Differences does not include a BaseEncoding entry, the interpretation of this is different depending on the original font Type. For Type 1 fonts, the built-in encoding refers to the Encoding of the font. For TrueType fonts the Encoding should be made up as described in the PDF specification version 1.6 in the section on Encoding for TrueType fonts on page 400. In particular after the BaseEncoding and Differences are applied any unassigned entries are assigned based on StandardEncoding (STD in the Encoding table) as defined in Appendix D.1 of the specification.

10.3.3 Schema For Font Descriptors

```
font_file =
    element Font { font_dictionary }
font_dictionary =
    element Type0 { type0_font_dictionary }
font_dictionary |=
    element Type1 { type1_font_dictionary }
font_dictionary |=
    element TrueType { truetype_font_dictionary }

common_font_dictionary =
      attribute Name { name }?  & attribute Name_enc { token }?
   & attribute BaseFont { name }  & attribute BaseFont_enc { token }?
   & attribute FirstChar { integer }?
   & attribute LastChar { integer }?
   & element Widths { array_of_integer }
```

```
    & element FontDescriptor { Font_descriptor_dictionary }

type0_font_dictionary =
     element DescendantFonts { array_of_1_CIDFontDictionaries }?
 & common_font_dictionary

type1_font_dictionary = common_font_dictionary
truetype_font_dictionary = common_font_dictionary

Font_descriptor_dictionary =
     attribute FontName { name }  & attribute FontName_enc { token }?
   & attribute FontFamily { pdf_byte_string }?    & attribute FontFamily_enc { token }?
   & attribute FontStretch { name }?  & attribute FontStretch_enc { token }?
   & attribute FontWeight { number }?
   & attribute Flags { ( list { "FixedPitch"? , "Serif"? , "Symbolic"? , "Script"? ,
"Nonsymbolic"? , "Italic"? , "AllCap"? , "SmallCap"? , "ForceBold"?  } ) }
   & attribute FontBBox { rectangle_blank_sep }
   & attribute ItalicAngle { number }
   & attribute Ascent { number }
   & attribute Descent { number }
   & attribute Leading { number }?
   & attribute CapHeight { number }
   & attribute XHeight { number }?
   & attribute StemV { number }
   & attribute StemH { number }?
   & attribute AvgWidth { number }?
   & attribute MaxWidth { number }?
   & attribute MissingWidth { number }?
   & attribute CharSet { pdf_byte_string }?    & attribute CharSet_enc { token }?
```

10.4 Font Permissions

Fonts embedded in the Mars file that can be used by forms during editing must have editable embedding permissions.

In general, font embedding permissions and guidelines should be closely followed by all applications in order to avoid copyright infringement. For details, see http://partners.adobe.com/public/developer/en/acrobat/sdk/FontPolicies.pdf.

10.5 Font Protection

Embedded fonts are obfuscated using the following algorithm. This prevents their use outside the document to which they are assigned.

Algorithm 1 ***Embedded Font Obfuscation***

1. Obtain the value of the DocumentID attribute in the document backbone. If this attribute is missing or the null string, embedded fonts cannot be used in the document.
2. Construct an obfuscation key. Use the DocumentID value as a BASE64-encoded representation of the obfuscation key. If the value is longer than 512 bits, only the first 512 bits are used.
3. XOR the first 1024 bytes of font data with the key by repeating the key value enough times so that its length matches the length of the font data or 1024 bytes, whichever is less. The font data is processed as a sequence of bits where each byte of data is interpreted as a sequence of eight bits with the high-order (most significant) bit listed first. The obfuscated data is stored in the package as the content of the font.

Font data is restored to its un-obfuscated state using the same algorithm.

11 Images

Mars represents images using standard formats. In general, all of the image information should be stored within the image file itself. The exceptions are use of "sidecar" files for XMP metadata and use of external color profiles.

When image data is converted between Mars and PDFC formats, lossless image encodings should convert exactly, and lossy image encodings should convert without introducing additional loss. This is an implementation guideline and not part of the file format definition.

Images are stored in separate files in the document package. These files may be in any non-reserved area of the package directory structure but conventionally should only appear in the /image directory or the page directory for the page on which the image is referenced.

The following image file types are supported in Mars:

TABLE 17 Supported Image Formats in Mars

FORMAT	NOTES
JPEG	
JPEG2000	Includes loss-less compression algorithms. Data used in Mars JPEG2000 images should be limited to the JPX baseline set of features, except for enumerated color space 19 (CIEJab) and enumerated color space 12 (CMYK) which are also supported.
JBIG2	
PNG	

Images in PDFC are stored in streams and compressed using one or more compression filters appropriate for the type of image data.

TABLE 18 Mars Image Format Correspondence to PDF

MARS IMAGE TYPE	PDF COMPRESSION FILTER	CONVERSION NOTES
JPEG	DCTDecode	Direct copy of image bytes
JPEG2000	JPXDecode	Direct copy of image bytes
JBIG2	JBIG2Decode	
PNG	other + indexed color	Construct image from image bytes plus image dictionary information
PNG	other + grayscale color	Construct image from image bytes plus image dictionary information
PNG	other + RGB color	Construct image from image bytes plus image dictionary information
JPEG2000	other + CMYK and/or spot color	

PDFC content stream inline images are not supported in Mars. They are converted to separate package-level image files.

12 Color Spaces

12.1 What Is A Color Space?

PDF includes powerful facilities for specifying the colors of graphics objects to be painted on a page. A PDF file can specify abstract colors in a device-independent way. Colors can be described in any of a variety of color systems, or color spaces. Some color spaces are related to device color representation (grayscale, RGB, CMYK), others to human visual perception (CIE-based).

Color spaces can be managed (profiled - based on some abstract model) or unmanaged (based on some device characteristics).

SVG (and hence Mars) favor the use of ICC profiles to represent color spaces. Since ICC profiles cannot cover all of the facilities defined by PDF, custom namespace extensions are defined for use by Mars documents to support other types of PDF-defined color spaces.

The following sections define how color spaces are specified and used in a Mars file, and the mapping between these extended color spaces and PDF color spaces.

12.2 Color Space Definition And Use

Within SVG page content, color spaces definitions appear in the document's <svg:defs> section. These definitions may refer to external ICC profile files or other relevant files. These external (to the SVG file) resources are contained in the document package. Color space definitions can also appear in resource files that appear elsewhere in the document package.

Colorspaces are defined via the svg <color-profile> element or one of several pdf colorspace type elements. Colorspaces that refer to an ICC color profile use the svg:color-profile element, with some Mars-specific additions depending on the specific color space.

The color space definitions in the SVG file must appear before they are used.

Color spaces are referenced in various elements and attributes defined by SVG, including the properties 'fill', 'stroke' 'stop-color', 'solid-color', 'flood-color' and 'lighting-color'. In addition, color spaces can reference other color spaces on which their definition depends.

References to color spaces themselves are generally done using a URI or simple color space name. When referenced using a URI, the color space definition can appear in a file other than the current SVG file. When referenced using a simple name, the color space must be defined in the <defs> element of the referencing SVG file.

All color values specified via ‘fill’ and ‘stroke’ properties include both the sRGB equivalent for the color value (as required by the SVG language specification) and a second function value which contains the color values that are actually to be used. For color spaces with an ICC profile, the icc-color(...) function is used. Here is an example of a path element which is filled with CMYK of (1, .5, .3, 0). Note the reference to the color space “cs-1” whose definition would include the reference to the ICC profile.

```
<path d="..." fill="rgb(0,113,150) icc-color(cs-1,1,.5,.3,0)”
```

Each <color-profile> element includes an "xlink:href" attribute. This value represents a unique URL that specifies the location of an ICC color profile. This approach applies to the following color spaces: CalGray, CalRGB, and L*a*b*.

For color spaces without an ICC profile, the color value is instead specified via the device-color(...) function. The format for this function is similar: a color-space identifier, followed by an n-tuple of color values, where n is determined by the number of color components in the color space. For the DeviceCMYK and DeviceGray colorspaces, which require no supplemental information to be stored in a color space element, the string "DeviceCMYK" or "DeviceGray" is used as the color space name and these names are considered pre-defined. A color space element and the device-color(...) function are used for Indexed, Pattern, DeviceN, and Separation color spaces.

Note: *icc-color(...) is part of the current SVG 1.1 specification; device-color(...) is slated to appear in SVG 1.2.*

```
fill="rgb(0,0,0) icc-color(cs-1,0.796,0.722,0.78,0.537)"
<image xlink:href="/...-38.png" width="252" height="239" color-profile="cs-3" />
```

12.3 Types Of Color Spaces and Their Representation

SVG supports sRGB color and ICC profiled color spaces. PDF defines a number of additional color spaces. Of these, CalRGB and Lab are represented as SVG ICC Profiles color spaces. The remainder are represented using PDF namespaced extensions to SVG. PDF Defines the following color spaces:

TABLE 19 Color Space Types

SPACE	DESCRIPTION	REPRESENTATION IN MARS
Device Color Spaces		
DeviceGray	Unmanaged, unprofiled, grayscale, no ICC equivalent. Controls the intensity of achromatic light, on a scale from black to white.	Predefined space name, “DeviceGray”. Reference: "rgb(rgb-equivalent) device-color(DeviceGray, intensity-value)" Example: fill="rgb(86,86,86) device-color(DeviceGray,.6625)"
DeviceRGB	Unmanaged, unprofiled, RGB based, no ICC equivalent. Controls the intensities of red, green, and blue light, the three additive primary colors used in displays.	DeviceRGB/sRGB is specified via the default 'rgb(r, g, b)' function present in every color specifier. For DeviceRGB, no additional icc-color or device-color function is needed. Reference: rgb(r-value, g-value, b-value) Example: fill="rgb(12,91,255)"
DeviceCMYK	Unmanaged, unprofiled, CMYK based, no ICC equivalent. Controls the concentrations of cyan, magenta, yellow, and black inks, the four subtractive process colors used in printing.	Predefined space name, “DeviceCMYK”. Reference: "rgb(rgb-equivalent) device-color(DeviceCMYK, c, m, y, k)" Example: fill="rgb(12,91,255) device-color(DeviceCMYK,.89,.62,0,0)"
CIE-based Color Spaces		

TABLE 19 Color Space Types

SPACE	DESCRIPTION	REPRESENTATION IN MARS
CalGray	Managed, grayscale. A special case of a single-component CIEbased color space, known as a CIE-based A color space. This type of space is the one-dimensional (and usually achromatic) analog of CIE-based ABC spaces.	pdf namespace extension to SVG.[a] CalGray_color_space_array = element CalGray { attribute Name { text }? & attribute id { text}? & CalGray_color_space_dictionary } CalGray_color_space_dictionary = attribute WhitePoint { tristimulus_xyz_triple } & attribute BlackPoint { tristimulus_xyz_triple }? & attribute Gamma { number }? tristimulus_xyz_triple = {number number number}
CalRGB	Managed, RGB based. A CIE-based ABC color space with only one transformation stage instead of two. In this type of space, A, B, and C represent calibrated red, green, and blue color values in the range 0.0 to 1.0.	pdf namespace extension to SVG.[b] <svg:color-profile xlink:href="icc profile ref"> Additional attributes defined: attribute pdf:WhitePoint { tristimulus_xyz_triple }? attribute pdf:BlackPoint { tristimulus_xyz_triple }? attribute pdf:Gamma { gamma_rgb_triple }? attribute pdf:Matrix { linear_interpretation_array } ? tristimulus_xyz_triple = number number number gamma_rgb_triple = number number number linear_interpretation_array = number number number number number number number number number
Lab	A CIE-based ABC color space with two transformation stages. In this type of space, A, B, and C represent the L*, a*, and b* components of a CIE 1976 L*a*b* space.	pdf namespace extension to SVG.[c] <svg:color-profile name="cs-2" xlink:href="icc profile ref"> Additional attributes defined: attribute pdf:WhitePoint { tristimulus_xyz_triple }? attribute pdf:BlackPoint { tristimulus_xyz_triple }? attribute pdf:Range { lab_color_ab_range }? lab_color_ab_range = number number number number

TABLE 19 Color Space Types

SPACE	DESCRIPTION	REPRESENTATION IN MARS
ICCBased	Managed, ICC profile based. Based on a cross-platform color profile as defined by the International Color Consortium (ICC). An ICCBased color space is characterized by a sequence of bytes in a standard format. Details of the profile format can be found in the ICC specification	Represented as an ICC profiled space with a few pdf namespace extensions: <svg:color-profile name="cs-2" xlink:href="icc profile ref"> Additional attributes defined: attribute pdf:Count {number }? // 1, 3, or 4 element Alternate { color_space }? attribute Range { array_of_number }?
Special Color Spaces		
Indexed	Adds a lookup table to one of the other color spaces (except Pattern). Allows page content to use small integers as indices into a color map or color table of arbitrary colors in some other space. This technique can considerably reduce the amount of data required to represent a sampled image—for example, by using 8-bit index values as samples instead of 24-bit RGB color values. No ICC equivalent	pdf namespace extension to SVG. indexed_color_space = element Indexed { attribute Name { text }? & attribute id { text }? & attribute Base { color_space_ref } & attribute HiVal { integer } & element LookupTable { lut_string_or_stream } lut_string_or_stream = pdf_base64_string lut_string_or_stream |= common_stream_dictionary

TABLE 19 Color Space Types

SPACE	DESCRIPTION	REPRESENTATION IN MARS

Pattern	A Pattern color space enables page content to paint an area with a pattern rather than a single color. The pattern may be either a tiling pattern (type 1) or a shading pattern (type 2).
	
	Type 1 and type 2 patterns can include color information in the pattern itself. Type 1 patterns can also be uncolored with any color coming from the point of reference to the pattern.
	
	Pattern spaces are modelled as color spaces in both PDF and Mars.
		pdf namespace extension to SVG.
		pattern_dictionary =
		element TilePattern {
		attribute Name { text }?
		& attribute id { text }?
		& type_1_pattern_dictionary }
		pattern_dictionary |=
		element ShadePattern {
		attribute Name { text }
		& attribute id { text }?
		& type_2_pattern_dictionary }
		
		type_1_pattern_dictionary =
		attribute PaintType { integer }
		& attribute TilingType { integer }
		& attribute BBox { rectangle }
		& attribute XStep { number }
		& attribute YStep { number }
		& attribute Matrix { coordinate_map_array }
		& common_stream_dictionary
		
		type_2_pattern_dictionary =
		attribute Shader { shading_dictionary_ref }
		& attribute Matrix { coordinate_map_array }?
		& element ExtGState {
		graphic_state_parameter_dictionary }?

Separation	Provides a means for specifying the use of additional colorants or for isolating the control of individual color components of a device color space for a subtractive device. When such a space is the current color space, the current color is a single-component value, called a tint, that controls the application of the given colorant or color components only. Supports “spot colors”.	pdf namespace extension to SVG. color_space_array |= element Separation { attribute Name { text }? & attribute id { text}? & separation_color_space } separation_color_space = attribute Colorant { name } & attribute AlternateSpace { color_space_ref } & element TintTransform { function_dictionary }

TABLE 19 Color Space Types

SPACE	DESCRIPTION	REPRESENTATION IN MARS
DeviceN	Can contain an arbitrary number of named color components and alternates should they be unavailable. They provide greater flexibility than is possible with standard device color spaces such as DeviceCMYK or with individual Separation color spaces. For example, it is possible to create a DeviceN color space consisting of only the cyan, magenta, and yellow color components, with the black component excluded. NChannel color spaces are also covered as part of DeviceN color spaces.	pdf namespace extension to SVG. color_space_array |= element DeviceN { attribute Name { text } & attribute id { text}? & device_n_color_space } device_n_color_space = element Colorants { colorant_dictionary } & attribute AlternateSpace { color_space_ref } & element TintTransform { function_dictionary } & element Attributes { device_n_color_space_attributes_dictionary }? device_n_color_space_attributes_dictionary = attribute Subtype { name }? & element Process { process_dictionary }? & element MixingHints { mixing_hints_dictionary }? process_dictionary = attribute ColorSpace { color_space_ref } & element Components { array_of_name } mixing_hints_dictionary = element Solidities { ink_density_dictionary } & element PrintingOrder { array_of_name } ink_density_dictionary = element Density { number & attribute Name { pdf_byte_string }}

a.In a future version, we would like to drop support for CalGray and convert to ICC profiles on conversion from PDF.
b.In a future version, we would like to drop support for CalRGB and convert to ICC profiles on conversion from PDF.
c.In a future version, we would like to drop support for Lab and convert to ICC profiles on conversion from PDF.

For more description of the characteristics of the PDF-defined color spaces, see the PDF 1.6 Reference Manual, section 4.5.

12.4 Examples of XML Specification and Use of Color Spaces

TABLE 20 Examples of Mars Color Spaces

COLOR SPACE TYPE		EXAMPLES
DeviceGray	Def	None
	Ref form	"rgb(rgb-equivalent) device-color(DeviceGray, intensity-value)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(86,86,86) device-color(DeviceGray,.6625)" stroke="rgb(255,255,255) device-color(DeviceGray,.9855)" />
DeviceRGB	Def	None
	Ref form	rgb(r-value, g-value, b-value)
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(86,23,86)" stroke="rgb(#F0A02B)" />
DeviceCMYK	Def	None
	Ref Form	"rgb(rgb-equivalent) device-color(DeviceCMYK, c, m, y, k)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(12,91,255) device-color(DeviceCMYK,.89,.62,0,0)" stroke="rgb(120,232,255) device-color(DeviceCMYK,.89,.62,4,34.5)" />
CalGray	Def	<svg:defs> <pdf:CalGray Name=”cs-1” pdf:WhitePoint=”1.0 1.0 1.0” pdf:BlackPoint=”0 0 0” pdf:Gamma=”23.4” /> ... </svg:defs> or in a separate resource file, /page/23/res.xml: <Resources> <CalGray id=”cs-23” WhitePoint=”1.0 1.0 1.0” BlackPoint=”0 0 0” Gamma=”23.4” /> ... </Resources>
	Ref Form	"rgb(rgb-equivalent) device-color(cs-id, intensity-value)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(86,86,86) device-color(cs-1,.6625)" stroke="rgb(86,86,86) device-color(url(/page/23/res.xml#cs-23),.6625)" />

TABLE 20 Examples of Mars Color Spaces

COLOR SPACE TYPE		EXAMPLES
CalRGB	Def	<svg:defs> <svg:color-profile name=”cs-8” xlink:href=”alphagraphics.icc” pdf:WhitePoint=”0 0 0” pdf:BlackPoint=”2 3 4” pdf:Gamma=”1 2 3” /> ... </svg:defs>
	Ref Form	"rgb(rgb-equivalent) icc-color(cs-id, r-value, g-value, b-value)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(86,86,86) device-color(cs-8,.6625)" stroke="rgb(86,86,86) device-color(url(/page/23/res.xml#cs-23),.6625)" />
Lab	Def	<svg:defs> <svg:color-profile name=”cs-2” xlink:href=”betaprints.icc” pdf:WhitePoint=”0 0 0” pdf:BlackPoint=”1 1 1” pdf:Range=”0 1 0 1” /> </svg:defs>
	Ref Form	"rgb(rgb-equivalent) icc-color(cs-id, l-value, a-value, b-value)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(246,177,153) icc-color(cs-2,80,20,20)" stroke="rgb(246,177,153) icc-color(cs-2,80,20,20)" />
ICCBased	Def	<svg:defs> <svg:color-profile name=”cs-9” xlink:href=”alphagraphics.icc” pdf:Count=” 3” pdf:Range=”0 1 0 1 0 1”/> ... </svg:defs>
	Ref Form	"rgb(rgb-equivalent) icc-color(cs-id, value1, value2, valuen)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(246,177,153) icc-color(cs-9,.4,.6,.1)" stroke="rgb(246,177,153) icc-color(cs-9,.4,.6,.1)" />

TABLE 20 Examples of Mars Color Spaces

COLOR SPACE TYPE		EXAMPLES
Indexed	Def	<svg:defs> ... <pdf:Indexed Name="cs-1" pdf:Base="cs-2" // cs-2 must be defined earlier pdf:HiVal="3" pdf:Gamma="23.4" > <pdf:LookupTable src="/color/cs-1.lut"> </pdf:Indexed> ... </svg:defs> or in a separate resource file, /page/p23/res.xml: <Resources> <Indexed id="cs-54" pdf:Base="cs-2" // cs-2 must be defined earlier pdf:HiVal="3" pdf:Gamma="23.4" > <LookupTable src="/color/cs-1.lut"> </Indexed> ... </Resources>
	Ref Form	"rgb(rgb-equivalent) device-color(cs-id, index)" ???
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(246,177,153) device-color(cs-1,34)" stroke="rgb(246,177,153) device-color(url(/page/23/res.xml#cs-54),2)" />

TABLE 20 Examples of Mars Color Spaces

COLOR SPACE TYPE		EXAMPLES
Pattern	Def	<svg:defs> ... <pdf:TilePattern Name=”pcs-1” pdf:PaintType=”1” pdf:TilingType=”1” src="/res/t1_3044082663-403.svg" pdf:BBox=”0 0 234 180” pdf:XStep=”2” pdf:YStep=”4” Matrix="-0.3 0 0 -0.3 365.625 414.375"> </pdf:TilePattern> ... </svg:defs> or in a separate resource file, /page/p23/res.xml: <Resources> ... <ShadePattern id=”pcs-23” Shader=”sha-32” Matrix="0.2 0.2 -0.2 0.2 323.3 725.9"> <ExtGState OverprintMode="1" Overprint="false" OverprintPaint="false" StrokeAdjust="false" SmoothnessTolerance="0.002"/> </ShadePattern> ... </Resources>
	Ref Form	Uncolored Type 1 Patterns: device-color(cs-id, val1, val2, ... valn) [where n is the number of components in the base colorspace] Colored Type 1 Patterns: device-color(cs-id) Type 2 Patterns: device-color(cs-id)
	Ref	"device-color(cs-1,.33,.25,0,1)" <circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(246,177,153) device-color(pcs-1)" stroke="rgb(246,177,153) device-color(url(/page/23/res.xml#pcs-23))" />

TABLE 20 Examples of Mars Color Spaces

COLOR SPACE TYPE		EXAMPLES
Separation	Def	<svg:defs> ... <pdf:Separation Name="cs-2" Colorant="ProcessBlack" AlternateSpace="cs-0"> <pdf:TintTransform> <pdf:SampledFunction Domain="0 1" Range="0 1 0 1 0 1" Size="255" BitsPerSample="8" Order="linear" Encode="0 254" Decode="0 1 0 1 0 1"> <pdf:File Name="/res/func-1.f0"/> </pdf:SampledFunction> </pdf:TintTransform> </pdf:Separation> ... </svg:defs>
	Ref Form	"rgb(rgb-equivalent) device-color(cs-id, tint-intensity-value)"
	Ref	<circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(246,177,153) device-color(scs-1, 0.7)" stroke="rgb(246,177,153) device-color(url(/page/23/res.xml#scs-54),0.5)" />
DeviceN	Def	<svg:defs> ... <pdf:DeviceN Name="cs-4" AlternateSpace="DeviceCMYK"> <pdf:Colorants> <pdf:Colorant Name="PANTONE 3282 CVC" Base=""/> </pdf:Colorants> <pdf:TintTransform> <pdf:SampledFunction Domain="0 1" Range="0 1 0 1 0 1 0 1" Size="255" BitsPerSample="8" Order="linear" Encode="0 254" Decode="0 1 0 1 0 1 0 1"> <pdf:File Name="/res/func-3.f0"/> </pdf:SampledFunction> </pdf:TintTransform> </pdf:DeviceN> ... </svg:defs> or in a separate resource file, /page/p23/res.xml: <Resources> ... <DeviceN id=”ncs-16” ... </DeviceN> </Resources>
	Ref Form	"rgb(rgb-equivalent) device-color(cs-id, val1, val2, ... valn)"
	Ref	"rgb(0,79,163) device-color(cs-1,.4,0,0,.75) <circle cx="600" cy="200" r="100" stroke-width="10" fill="rgb(0,79,163) device-color(ncs-1,.4,0,.75)" stroke="rgb(246,177,153) device-color(url(/page/23/res.xml#ncs-16),.4,0,.7)" />

Reference: http://www.color.org/ICC1V42old.pdf

13 Annotations

Annotations are split into two groups: content annotations with are explicitly referenced by the page information file and represent material that is considered a standard part of page content. This group include form fields and link annotations, for example.

The second group is markup annotations which represent human-reader created marks on the page as part of some review and approval cycle. These are implicitly associated with the page and document and can be added without changing any other files in the document.

Mars annotations can include an appearance stream. The appearance stream provides an SVG representation of the rendering of the annotation. While appearance streams are optional in PDF and Mars, PDF producers nearly always include them, for at least these reasons:

- There is no other source of the appearance for certain annotations, such as rollover buttons, custom stamps, signatures, etc.
- Annotations are considered to be interactive features that may not be supported in a non-interactive Mars consumer, such as a printer. This is especially important in the case of static appearances for dynamic annotations, such as multimedia or 3D.
- In general, the appearance produced by an annotation handler is not prescribed by PDF but is implementation-dependent. For instance, formatting of text in a text annotation is implementation-dependent. It is sometimes important to capture the actual appearance that the user sees in a final-form representation, for legal or archival purposes.

There are specific features that call for appearances to be rendered, such as NeedsAppearances for widget annotations (AcroForm fields).

Coordinates used in XML representing annotations are based on SVG-style page coordinates with the origin in the upper left corner of the page.

13.1 Markup Annotations

In addition to the PDF-defined annotations, there is one additional annotation tag, <Custom> that is used to represent annotations within PDF files whose type is not recognized.

TABLE 21 Mars Markup Annotation Examples

Text	<Text CreationDate="D:20060406125454-07'00'" Rect="36.993 55.447998 56.993 37.447998" Name="86370bc7-c2bf-46cf-8f3a-bdc815963d03" Color="#FFFF00" Flags="Print NoZoom NoRotate" ModDate="D:20060407161407-07'00'" IconName="Comment" Title="hli" Subj="Note"> <Contents>Note Tool </Contents> <RTContents>...;p>Note Tool </p></body> </RTContents> <Popup Rect="17.9999 180.598999 197.999 60.599976" Flags="Print NoZoom NoRotate" Open="false"/> <Appearance> <Down> <Graphic src="/page/0/form_1489951617-7.svg" Matrix="1 0 0 1 0 0" BBox="0 0 20 18"> </Graphic> </Down> <Normal> <Graphic src="/page/0/form_1489951617-12.svg" Matrix="1 0 0 1 0 0" BBox="0 0 20 18"> </Graphic> </Normal> </Appearance> </Text>

TABLE 21 Mars Markup Annotation Examples

FreeText	<FreeText CreationDate="D:20060406130739-07'00'" Rect="156.308 90.909973 275.7 73.192993" Name="1498d0df-51f3-4e44-8964-3a39a56b5c15" Color="#FFFF00" StartPoint="160.8,718.8" KneePoint="194.4,708.995" EndPoint="161.9,708.995" Flags="Print" ModDate="D:20060406130904-07'00'" Intent="FreeTextCallout" Title="hli" DefaultAppearance="0 0 0 rg /Helv 9 Tf" DefaultStyle="font: Helvetica 9.0pt; text-align:left; color:#000000 " Subj="Text Box" Fringe="38.0917 0.5002 0.5002 2.4064"> <Contents>callout tool</Contents> <RTContents>... callout tool </body> </RTContents> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-47.svg" Matrix="1 0 0 1 -156.307999 701.090027" BBox="156.308 701.09 275.7 718.807"> </Graphic> </Normal> </Appearance> </FreeText>
Line	<Line CreationDate="D:20060406130113-07'00'" Rect="386.7 51.6 430.7 40.8" Name="9c1f0f1f-874a-4ba7-99fb-dd561c030e47" Color="#FF0000" Flags="Print" Start="392.2,46.3" End="425.2,46.3" ModDate="D:20060406153457-07'00'" Title="hli" Subj="Line"> <Popup Rect="612 165.6 792 45.6" Flags="Print NoZoom NoRotate" Open="false"/> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-42.svg" Matrix="1 0 0 1 -386.75 740.15" BBox="386.749 740.15 430.749 751.15"> </Graphic> </Normal> </Appearance> </Line>

TABLE 21 Mars Markup Annotation Examples

Square	<Square CreationDate="D:20060406130041-07'00'" Rect="310.8 56.5 346.782 37.56" Name="bba6b782-cee8-418e-8804-91a4b995185b" Color="#FF0000" Flags="Print" ModDate="D:20060406153452-07'00'" Title="hli" Subj="Rectangle" Fringe="0.5, 0.5, 0.5, 0.5"> <Popup Rect="612 161.4 792 41.4" Flags="Print NoZoom NoRotate" Open="false"/> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-40.svg" Matrix="1 0 0 1 -310.832 735.436" BBox="310.832 735.436 346.782 754.436"> </Graphic> </Normal> </Appearance> </Square>
Circle	<Circle CreationDate="D:20060406130046-07'00'" Rect="358.1 55. 385.5 36.3" Name="911c0f0d-a9f9-45bb-9095-07748590a49b" Color="#FF0000" Flags="Print" ModDate="D:20060406153455-07'00'" Title="hli" Subj="Oval" Fringe="0.5, 0.5, 0.5, 0.5"> <Popup Rect="612 162 792 42" Flags="Print NoZoom NoRotate" Open="false"/> </Circle>
Polygon	<Polygon CreationDate="D:20060406130230-07'00'" Rect="479 52 520 35" Name="c4da25de-7770-48a8-9d10-a9eb8dddaaea" Color="#FF0000" Flags="Print" ModDate="D:20060406153507-07'00'" Title="hli" Subj="Polygon" Path="485.25,36;514.5,36;519.75,45;514.5,51;480.75,51"> <Popup Rect="612 156 792 36" Flags="Print NoZoom NoRotate" Open="false"/> </Polygon>
PolyLine	<PolyLine Rect="440. 49.75 473.5 37.25" Name="7ede4730-f13b-47bc-a81d-bbef8216beb5" Color="#FF0000" Flags="Print" Title="hli" Subj="Polygonal Line" Path="450,38.25;472.5,38.25;462.752,48.75;441,48.75;447.002,43.5;"> <Popup Rect="612 157.8 792 37.8" Flags="Print NoZoom NoRotate" Open="false"/> </PolyLine>
Highlight	<Highlight Rect="221.2 294.53 249.37 282.67" Name="6e1bb9d7-c823-448b-92f0- 06089ab3b739" Color="#FFFF00" Flags="Print" Title="hli" Subj="Highlight" Coords="224.17,283.02 246.39,283.02 224.17,294.18 246.39,294.18"> <Popup Rect="612 403.02 792 283.02" Flags="Print NoZoom NoRotate" Open="false"/> </Highlight>
Underline	<Underline CreationDate="D:20060406125734-07'00'" Rect="196.662 314.88 228.878 302.32" Name="c109c22a-e4bb-4089-8da1-91b3e5094c55" Color="#00FF00" Flags="Print" ModDate="D:20060406125734-07'00'" Title="hli" Subj="Underline" Coords="199.99,303.02 225.55,303.02 200,314.18 225.55,314.18"> <Popup Rect="612 423.02 792 303.02" Flags="Print NoZoom NoRotate" Open="false"/> </Underline>

TABLE 21 Mars Markup Annotation Examples

Squiggly	<Squiggly CreationDate="D:20060406125751-07'00'" Rect="269.09 314.68 310.621 302.52" Name="e65982bf-f86e-4f70-ad2e-70e425fd261a" Color="#00FF00" Flags="Print" ModDate="D:20060406125801-07'00'" Title="hli" Subj="Underline" Coords="272.22,303.02 307.49,303.02 272.22,314.18 307.49,314.18"> <Popup Rect="612 423.02 792 303.02" Flags="Print NoZoom NoRotate" Open="false"/> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-32.svg" Matrix="1 0 0 1 -269.089 477.32" BBox="269.09 477.32 310.621 489.48"> </Graphic> </Normal> </Appearance> </Squiggly>
StrikeOut	<StrikeOut CreationDate="D:20060406125743-07'00'" Rect="231.662 314.877991 263.878 302.321991" Name="7dc50503-7258-4496-a4be-40cdd0de3f81" Color="#FF0000" Flags="Print" ModDate="D:20060406125743-07'00'" Title="hli" Subj="Cross-Out" Coords="234.99,303.019989 260.55,303.019989 234.99,314.179993 260.55,314.179993"> <Popup Rect="612 423.019989 792 303.019989" Flags="Print NoZoom NoRotate" Open="false"/> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-31.svg" Matrix="1 0 0 1 -231.662003 477.122009" BBox="231.662 477.122 263.878 489.678"/> </Normal> </Appearance> </StrikeOut>
Caret	<Caret CreationDate="D:20060406125706-07'00'" Rect="162.977 316.450989 168.222 310.066986" Name="73b21f22-ed4f-4669-b4b8-7c03aa0cc0cd" Color="#0000FF" Flags="Print" ModDate="D:20060406125721-07'00'" Title="hli" Subj="Inserted Text" Fringe="0.9122 0.9122 0.9122 0.9122"> <Contents>Insert Text At Cursor</Contents> <RTContents><?xml version="1.0"?><body xmlns="http://www.w3.org/1999/xhtml" xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/" xfa:APIVersion="Acrobat:7.0.0" xfa:spec="2.0.2"> Insert Text At Cursor </body></RTContents> <Popup Rect="74.3998 295.799011 254.399 175.799011" Flags="Print NoZoom NoRotate" Open="false"/> </Caret>

TABLE 21 Mars Markup Annotation Examples

Stamp	<Stamp CreationDate="D:20060406125819-07'00'" Rect="72.04 53.927 170.049 33.9" Name="9d79feba-e194-4f07-9419-d9ca13a54d81" Color="#FF0000" Flags="Print" ModDate="D:20060406130615-07'00'" IconName="#DApproved" Title="hli" Subj="Approved"> <Popup Rect="612 156.487 792 36.487" Flags="Print NoZoom NoRotate" Open="false"/> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-34.svg" Name="FRM" Matrix="1 0 0 1 -180.22 369.479" BBox="180.22 369.479 433.857 421.305"> <Attributes Isolated="false" Knockout="false" Type="Transparency"> <ColorSpace>DeviceRGB</ColorSpace> </Attributes> </Graphic> </Normal> </Appearance> </Stamp>
Ink	<Ink CreationDate="D:20060406130441-07'00'" Rect="470.827 90.426025 498.66 72.984009" Name="23515430-c902-4c46-9990-6a86a041ac99" Color="#FF0000" Flags="Print" ModDate="D:20060406153105-07'00'" Title="hli" Subj="Pencil"> <Inklist> <Gesture Points="484.254,82.939026;483.756,83.258972;"/> <Gesture Points="491.736,77.822021; 492.734,78.142029; 493.232,78.781982; 493.731,78.781982; 493.731,79.101990; 494.23,79.421997; 494.23,80.060974;"/> <Gesture Points="472.781,76.862976;472.781,76.541992; 473.28,76.223022; 473.779,75.903015; 474.278,75.903015; 474.776,75.583008;475.774,75.263000; 477.27,74.624023;478.767,74.304016; 479.266,74.304016; 480.264,73.984009; 480.762,73.984009;"/> </Inklist> <Popup Rect="612 152.400024 792 32.400024" Flags="Print NoZoom NoRotate" Open="false"/> </Ink>

TABLE 21 Mars Markup Annotation Examples

FileAttachment	<FileAttachment CreationDate="D:20060406131253-07'00'" Rect="125.548 89.750000 132.548 72.750000" Name="6d695573-63a4-4afb-9601-b8de0bb874c6" Color="#3F54FF" Flags="Print NoZoom NoRotate" ModDate="D:20060406153101-07'00'" IconName="Paperclip" Title="hli" Subj="File Attachment"> <Contents>0_newsltr.pdf</Contents> <RTContents>...t; > 0_newsltr.pdf </body></RTContents> <File Name="0_newsltr.pdf"> <FileData src="/file/ef_1489951617-51" FileType="application/pdf" UncompressedSize="247271"> <Params CreationDate="D:20060404160321-07'00'" ModDate="D:20041028121435-07'00'" Size="247271" Checksum="NcqUVHTbUXE0veH3OSlxJA==/> </File> </File> </FileAttachment>
Sound	<Sound Rect="210.417 210.187988 223.417 192.187988" CreationDate="D:20030226121614" Name="KvgXBj-7NwX9LJruCFiQOC" Color="#FFFF00" Flags="Print NoZoom NoRotate" ModDate="D:20030226121836-08'00'" IconName="Mic" Title="jcanepa" Subj="Sound Attachment"> <Contents>Recorded Sound Clip (874 KB)</Contents> <SoundFile src="/media/sound_1489951617-52" BitsPerSample="16" ChannelCount="2" Encoding="Signed" SamplingRate="21436"> </SoundFile> <RTContents><body xmlns="http://www.w3.org/1999/xhtml" xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/" xfa:APIVersion="Acrobat:6.0.0" xfa:spec="2.0.2" > Recorded Sound Clip (874 KB) </body></RTContents> </Sound>
Custom	<Custom Rect="210.417 210.187988 223.417 192.187988" CreationDate="D:20030226121614" Name="KvgXBj-7NwX9LJruCFiQOC" Color="#FFFF00" Flags="Print NoZoom NoRotate" ModDate="D:20030226121836-08'00'"> ... </Custom>

13.2 Content Annotations

TABLE 22 Mars Content Annotation Examples

Link	<Link Rect="300.5 471 435.5 453.500000" Color="#FFA900" Highlight="Invert"> <Border Style="Solid" Width="1"/> <OnActivation> <Goto3DView> <View Index="0"/> <Target ref="/page/2/pg.can#0"/> </Goto3DView> </OnActivation> </Link>
Movie	<Movie Rect="407.999 223.200012 570.6 112.552002" Title="Annotation from TACODOG.avi"> <Border Style="Solid" Width="1"/> <File Name="TACODOG.avi"/> <Poster> <Image src="/page/3/im_1489951617-127" BitsPerComponent="8" Width="308" Height="210"> <ColorSpace>DeviceRGB</ColorSpace> </Image> </Poster> <Aspect Width="163" Height="111"/> </Movie>

TABLE 22 Mars Content Annotation Examples

Screen	<Screen Rect="114.751 214.25 272.75 110.75" Flags="Print" Title="Annotation from TACODOG.avi" ID="1"> <Border Style="Solid" Width="1"/> <OnActivation> <Rendition Operation="PlayAndAssociate"> <MediaRendition> <Media Name="Rendition from TACODOG.avi"> <Clip ContentType="video/avi" Name="Media clip from TACODOG.avi" Subtype="MCD"> <Data Name="TACODOG.avi"> <File src="/file/ef_1489951617-18" FileType="video/avi"> <Params CreationDate="D:20060405143633-07'00'" ModDate="D:20060405143634-07'00'" Size="771558" Checksum="oVN6s7ZSVCYSzorCn98B7g=="/> </File> </Data> <Permissions TempFile="TEMPACCESS"/> </Clip> </Media> </MediaRendition> <Screen ref="#1"/> </Rendition> </OnActivation> <AppearanceCharacteristics> <Icon src="/page/0/form_1489951617-57.svg" Matrix="1 0 0 1 0 0" BBox="0 0 1 1"/> </AppearanceCharacteristics> <Appearance> <Normal> <Graphic src="/page/0/form_1489951617-59.svg" BBox="0 0 157.999 103.5"/> </Normal> </Appearance> </Screen>

TABLE 22 Mars Content Annotation Examples

Widget	<Widget Rect="531.002 53.250000 568.502 34.500000" Flags="Print" AppearanceState="Off" ID="0"> <AppearanceCharacteristics BackgroundColor="1" BorderColor="0 0.6667 0"/> <Appearance> <Down> <Graphics> <Graphic src="/page/0/form_1489951617-53.svg" StateName="Yes" Matrix="1 0 0 1 0 0" BBox="0 0 37.5 18.75"> </Graphic> <Graphic src="/page/0/form_1489951617-54.svg" StateName="Off" Matrix="1 0 0 1 0 0" BBox="0 0 37.5 18.75"> </Graphic> </Graphics> </Down> <Normal> <Graphics> <Graphic src="/page/0/form_1489951617-55.svg" StateName="Yes" Matrix="1 0 0 1 0 0" BBox="0 0 37.5 18.75"> </Graphic> <Graphic src="/page/0/form_1489951617-56.svg" StateName="Off" Matrix="1 0 0 1 0 0" BBox="0 0 37.5 18.75"> </Graphic> </Graphics> </Normal> </Appearance> </Widget>
PrinterMark	

TABLE 22 Mars Content Annotation Examples

TrapNetwork	<TrapNet Rect="0 792 612 0" Flags="Print ReadOnly" AppearanceState="DefaultStyle"> <Version> <Obj ref="/page/1/info.xml#0" ref_type="8"/> <Obj ref="/page/1/res.xml#7" ref_type="6"/> <Obj ref="/page/1/res.xml#2" ref_type="6"/> </Version> <Appearance> <Normal> <Graphics> <Graphic src="/page/1/form_1489951617-83.svg" StateName="DefaultStyle" PCM="DeviceCMYK" Name="TFM0" Matrix="1 0 0 1 0 0" TrapStyles="DefaultStyle" BBox="0 0 612 792"> <SeparationColorNames> <Name Value="Cyan"/> <Name Value="Magenta"/> <Name Value="Yellow"/> <Name Value="Black"/> </SeparationColorNames> </Graphic> </Graphics> </Normal> </Appearance> </TrapNet>
Watermark	<Watermark Rect="157.332 303.971893 274.668 272.028107" Flags="Print ReadOnly"> <ContentGroup> <Membership> <Groups> <Group ref="/backbone.xml#1"/> </Groups> </Membership> </ContentGroup> <FixedPrint HorizontalTranslate="0.5" VerticalTranslate="0.5" Matrix="1 0 0 1 -58.668106 159.972000"/> <Appearance> <Normal> <Graphic src="/page/2/form_1489951617-97.svg" Matrix="1 0 0 1 0 0" BBox="0 0 117.336 31.9438"> </Graphic> </Normal> </Appearance> </Watermark>

TABLE 22 Mars Content Annotation Examples

3D	<A3D Rect="0 288 432 0" Name="3D1" Flags="Print ReadOnly" ID="0"> <Contents>3D Model</Contents> <Activation ArtworkOff="Instantiate"/> <Artwork> <Stream src="/annot/3D_1489951617-94"> <Views> <View InternalName="Unnamed" CenterOrbitDistance="4986.6099" Camera2World="-0.3827 0.9239 0 0.1802 0.0747 0.9808 0.9061 0.3753 -0.1951 -4574.25 -1626.22 1541.41" ExternalName="start" CameraTransformation="M"> <Projection Subtype="Perspective" FieldOfView="30" ProjectionViewBox="Min"/> </View> </Views> <AnimationStyle TimeMultiplier="1" Style="Linear" PlayCount="-1"/> </Stream> </Artwork> <Initial ID="1"> <View ref="/page/2/pg.can#1"/> </Initial> <Appearance> <Normal> <Graphic src="/page/2/form_1489951617-95.svg" Matrix="1 0 0 1 0 0" BBox="0 0 432 288"> </Graphic> </Normal> </Appearance> </A3D>

14 Actions

PDF supports a set of predefined actions and JavaScript actions. Predefined actions have specific Mars markup to represent them. JavaScript actions represent the execution of a script. The JavaScript code can appear inline in a <JavaScript> element or the <JavaScript> element can reference a separate package level file which contains the JavaScript source code.

JavaScript actions can appear in several places in a Mars document including anywhere a predefined action can appear and as a child of various event elements indicating what should happen for specific document and form-related events.

XML forms also define a rich JavaScript programming environment. JavaScripts related to XML forms are independent of document-related JavaScript actions and AcroForm (non-XML form) related JavaScript actions.

In addition to the predefined action types, there is a custom tag, <CustomAction> that is used to represent action types from a PDF file that is not recognized.

Predefined actions include the following:

TABLE 23 Predefined Actions

ACTION	DESCRIPTION	EXAMPLE MARKUP
GoTo	Go to a destination in the current document.	<GoTo> <Dest> <Fit Page_ref="/backbone.xml#p34"/> </Dest> </GoTo>
GoToR	(“Go-to remote”) Go to a destination in another document.	
GoToE	(“Go-to embedded”; PDF 1.6) Go to a destination in an embedded file.	
Launch	Launch an application, usually to open a file.	<Launch NewWindow="true"> <File Name="Fonts.pdf"/> </Launch>
Thread	Begin reading an article thread.	<Thread> <ArticleThread ref="/articles.xml#0"/> </Thread>
URI	Resolve a uniform resource identifier.	<URI URI="http://www.adobe.com"></URI>
Sound	Play a sound.	<Sound> <SoundFile src="/media/sound_1489951617-67" Encoding="muLaw" SamplingRate="8000"> </SoundFile> </Sound>
Movie	Play a movie.	<Movie Title=”Annotation from TACODOG” />
Hide	Set an annotation’s Hidden flag.	<Hide Hide="false"> <Target> <Field Pathname="Go to a 3D view"/> </Target> </Hide>

TABLE 23 Predefined Actions

ACTION	DESCRIPTION	EXAMPLE MARKUP
Named	Execute an action predefined by the viewer application.	<Named Name="PageSetup"/>
SubmitForm	Send data to a uniform resource locator.	<SubmitForm> <File Name="http://a...dobe.com/FormsTestSui#FDF" FSType="URL"/> </SubmitForm>
ResetForm	Set fields to their default values.	<ResetForm Flags="Exclude"> <Fields> <Field Pathname="Check Box3"/> <Field Pathname="List Box"/> <Field Pathname="Ping Result"/> <Field Pathname="Signature10"/> <Field Pathname="Text9"/> </Fields> </ResetForm>
ImportData	Import field values from a file.	<ImportData> <File Name="/E//hli/Desktop/File1_data.fdf"/> </ImportData>
JavaScript	Execute a JavaScript script.	<Script>AFPercent_Format(3, 0);</Script>
SetOCGState	Set the states of optional content groups.	<SetOCGState PreserveRB="false"> <State> <Group ref="/backbone.xml#2" Mode="OFF"/> <Group ref="/backbone.xml#5" Mode="OFF"/> <Group ref="/backbone.xml#1" Mode="ON"/> <Group ref="/backbone.xml#3" Mode="ON"/> </State> </SetOCGState>

TABLE 23 Predefined Actions

ACTION	DESCRIPTION	EXAMPLE MARKUP
Rendition	Controls the playing of multimedia content.	<Rendition Operation="PlayAssociated"> <MediaRendition> <Media Name="Rendition from ....avi"> <Clip ContentType="video/avi" Name="Media clip from ....avi" Subtype="MCD"> <Data> <File Name="TACODOG.avi"> <FileData src="/file/ef_1489951617-18" FileType="video/avi" UncompressedSize="771558"> <Params CreationDate="D:20060..."' ModDate="D:200604..."' Size="771558" Checksum="oVN6sSVCYS..."/> </FileData> </File> </Data> <Permissions TempFile="TEMPACCESS"/> </Clip> </Media> </MediaRendition> <Screen ref="/page/0/pg.can#1"/> </Rendition>
Trans	Updates the display of a document, using a transition parameters.	<Trans Style=”Cover” Direction=”90” />
GoTo3DView	Set the current view of a 3D annotation	<Goto3DView> <View Index="0"/> <Target ref="/page/2/pg.can#0"/> </Goto3DView>

Example 15 Mars Markup for Events and Actions

```
<Actions>
    <OnFocusIn>
        <URI URI="http://www.adobe.com"></URI>
    </OnFocusIn>
    <OnCursorEnter>
        <Hide Hide="false">
            <Target>
```

```
            <Field Pathname="CheckBox2"/>
          </Target>
        </Hide>
      </OnCursorEnter>
      <BeforeFormat><Script>AFPercent_Format(3, 0);</Script></BeforeFormat>
    </Actions>
```

15 Metadata

The XMP schema is used for metadata, which can be associated with the document level and a number of other places in PDF. The format of XMP data is specified at http://partners.adobe.com/asn/developer/xmp/pdf/MetadataFramework.pdf.

In Mars, the XMP metadata does not include any padding or binary header or trailer.

Document level metadata is always placed in the file "/META-INF/metadata.xml". In other places, implicit association is used to connect metadata to a files in the Mars document package. That is, if a file "/x/y/z" is present in the package, metadata associated with it is in the file "/x/y/z.xmp". This allows metadata to be added without necessarily modifying the document in other ways. Metadata can be associated with pages, fonts, images, page fragments (form xobjects), and ICC color profiles.

Metadata always contains supplementary information and never contains information that affects the appearance of the document.

The XMP wrapper packet is not used in Mars.

16 Marked Content & Logical Structure

Marked-content operators (PDF 1.2) identify a portion of content within a PDF content stream as being of interest to an application or plug-in. Property lists can be associated with the marked content.

The logical structure facilities of PDF (PDF 1.3) provide a mechanism for incorporating structural information about a document's content into a PDF file. Such information might include, for example, the organization of the document into chapters and sections or the identification of special elements such as figures, tables, and footnotes. The logical structure facilities are extensible, allowing applications that produce PDF files to choose what structural information to include and how to represent it, while enabling PDF consumers to navigate a file without knowing the producer's structural conventions. The logical structure mechanism in PDF uses marked-content operators to identify the content associated with it.

Tagged PDF (PDF 1.4) is a stylized use of PDF that builds on the logical structure framework. It defines a set of standard structure types and attributes that allow page content (text, graphics, and images) to be extracted and reused for other purposes. It is intended for use by tools that perform the following types of operations: Simple extraction of text and graphics for pasting into other applications, Automatic reflow of text and associated graphics, Processing text for such purposes as searching, indexing, and spell-checking, accessibility, and conversion to other common file formats (such as HTML, XML, and RTF).

16.1 Marked-Content

The marked-content mechanism in PDF allows any content within a content stream to be grouped. It can also be used in a non-grouping mechanism for adding extra information for a point in the content stream.

Mars extends the SVG <g> element using attributes in the PDF namespace to provide this capability. A <g> element using this mechanism is referred to as a marked content element. All elements and content enclosed within it are part of the marked-content.

An example of a marked content element is show below.

```
<g pdf:Mark="MyName" pdf:Prop_ref="Prop1">
    ... SVG content
</g>

or, for an inline property set


<g pdf:Mark="MyName">
   <pdf:Props xml:id="Prop1">
```

```
        ...
    </pdf:Props>
     ... SVG content
</g>
```

For a group element to be considered “marked”, it must contain the pdf:Mark attribute. The value of the Mark attribute is not unique and can be shared both within the same page and across multiple pages. Marked points are represented using the pdf namespace element <pdf:Point>. Marked points may have an optional <pdf:Props> child element.

Optionally, two mechanisms can be used to add extra information to the marked content element.

The first mechanism allows extra properties to be set inline within the SVG. This is shown in the second example above through the use of the <pdf:Props> element. The contents of this are not restricted, but must use the standard mechanism in Mars for specifying extension content (See “Representing PDFC Extensions” on page 131). If an inline property is to be used more than once, an xml:id can be associated with the property and that can later be re-used by referencing it with the pdf:Prop_ref attribute.

The second mechanism is to include a reference to an already existing property set. This can be in another file and referenced via a URI (/page/i/properties.xml#Prop1) or it can be a reference to property in the same file. This allows multiple marked content elements to share properties. The first example above shows this.

16.2 Structure Tree Introduction

The PDF structure tree provides a structured view of the otherwise linear document content. The page content is broken into chunks using the marked-content mechanism described above. The tree defines the order in which the content should flow and the role and type of each content element.

Leaf nodes in the structure tree point to marked content elements on the pages of the document and provide information about that content element such as:

- Language: the language in which the element is expressed.
- Alternate Text: a description of the element, typically applied to figures and images, for accessibility purposes
- Actual Text: a Unicode representation of the actual text content
- Class (such as paragraph, heading, table cell, etc.)
- Attributes: detailed parameters about the spacing, layout, or other characteristics of the content.

Most leaf node information is stored in the page contents file.

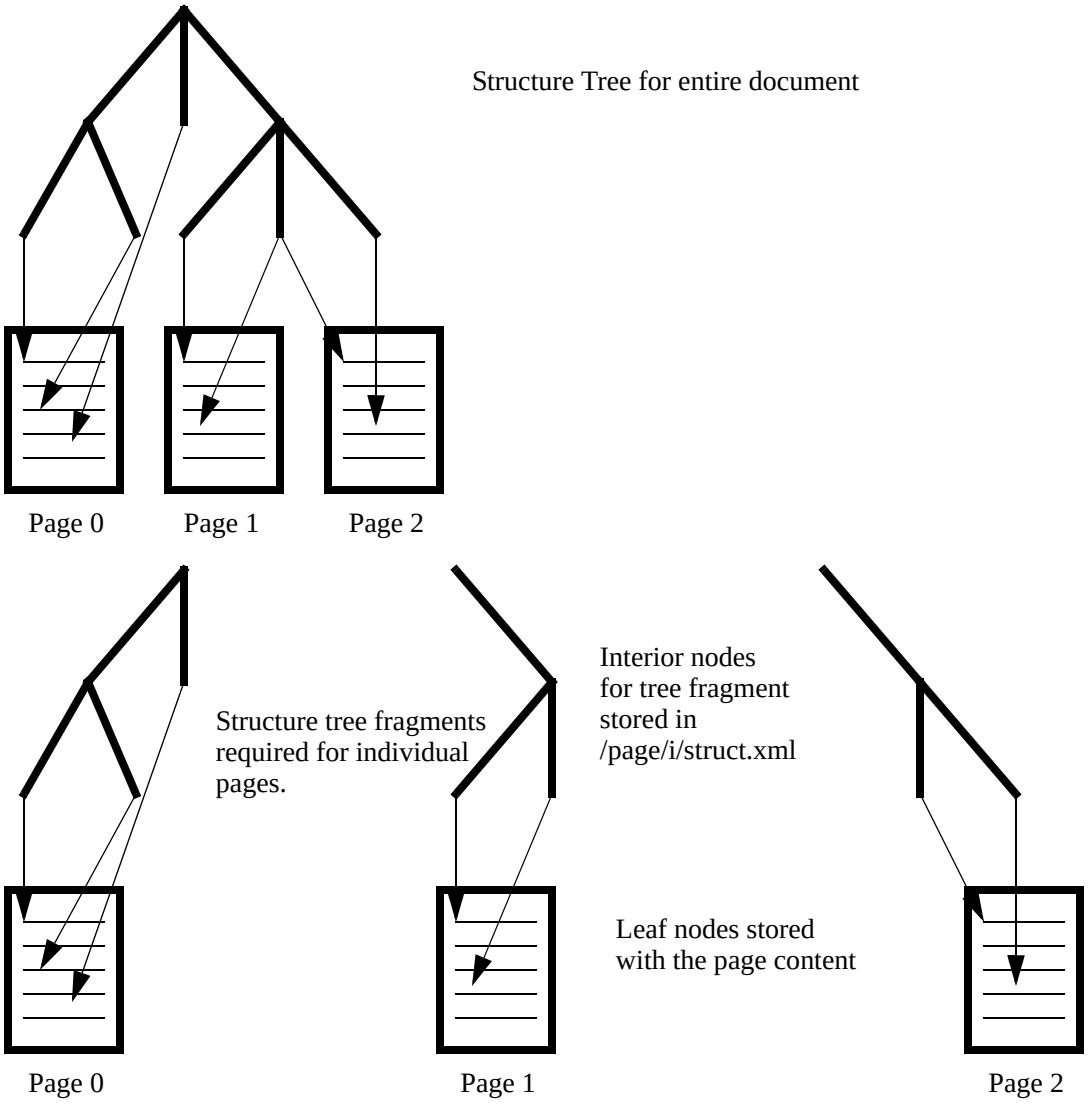
Interior (non-leaf) nodes of the tree provide information about groups of content elements and their relationship. Structures such as tables which consist of a heading, body, rows, and cells result in a number of interior structure tree nodes. Chapters, sections, and figures are also represented by subtrees in the structure tree. Interior structure nodes can contain the same information listed above for leaf elements; most often, they have class and attribute information.

To indicate that a document has structure, the element <Marked> appears in the document backbone. This element can also flag other special properties of the document.

Example 16 Element indicating that document is structured

```
<Marked TaggedPDF="true"/>
```

FIGURE .1 *Structure Tree Breakdown in Mars*



16.3 Structure Tree Breakdown in Mars

In Mars, the structure tree is not represented as a single tree but is broken into separate pieces of information that are stored in different locations in the document package.

16.3.1 Role and Class Maps

The Role Map and Class Map are stored in the document backbone. The Role map maps author defined structure tags to standard structure tags. The Class map defines default attribute values for particular classes of attributes. The role map and class map are described in the document backbone schema.

Example 17 Class Map as it appears in the <PDF> element of the document backbone

```
<StructureTypes>
    <ClassMap>
        <Class Name="CM1">
            <Attribute Owner="Layout">
                <TextAlign type="name" Value="Center"/>
            </Attribute>
        </Class>
        <Class Name="CM2">
            <Attribute Owner="Layout">
                <TextAlign type="name" Value="None"/>
                <LineHeight type="integer" Value="12"/>
            </Attribute>
        </Class>
    </ClassMap>
</StructureTypes>
```

16.3.2 Structure Per Page

In each page structure file is a series of entries for nodes of the structure tree that either directly or indirectly contain items of content belonging to that page. Both interior nodes (structure element) and the leaf content nodes (marked content element reference) are described within this file. Each <Elem> describes an interior node in the structure tree and contains a “Tag” attribute representing the name of the tag and “Role” attribute indicating the role-mapped tag of the content subtree. Each marked content element reference is described by an <MCR> element with a ref attribute referencing the page containing the content and the specific

marked content element (using a fragment identifier in the ref to name the id of the marked content element).

Example 18 Page Structure File for Interior Structure Tree Nodes

```
<Structure>
    <Elem Path="/1" Role="Sect" Tag="Sect"/>
    <Elem Path="/1/1" Pg="/page/0/info.xml" Role="P" Tag="Heading 1"/>
    <Elem ActualText="Global Electronics Logo" Alt="Global Electronics Logo"
          Path="/1/1/1" Pg="/page/0/info.xml" Role="Figure" Tag="InlineShape"/>
    <MCR Path="/1/1/1/1" ref="/page/0/pg.svg#MCID_0"/>
    <Elem Path="/1/2" Pg="/page/0/info.xml" Role="H1" Tag="H1"/>
    <MCR Path="/1/2/1" ref="/page/0/pg.svg#MCID_3"/>
    <Elem Path="/1/3" Pg="/page/0/info.xml" Role="P" Tag="P"/>
    <MCR Path="/1/3/1" ref="/page/0/pg.svg#MCID_4"/>
    <Elem Path="/1/4" Pg="/page/0/info.xml" Role="H1" Tag="H1"/>
    <MCR Path="/1/4/1" ref="/page/0/pg.svg#MCID_6"/>
    <Elem Path="/1/5" Pg="/page/0/info.xml" Role="P" Tag="P"/>
    <MCR Path="/1/5/1" ref="/page/0/pg.svg#MCID_7"/>
</Structure>
```

All <Elem> and <MCR> entries must also have a Path associated with them to describe their placement in the structure tree. The mechanism for this is described in the next major section. The schema for the per-page structure file follows.

```
structure_file =
      element Structure { structure_file_contents *}
structure_file_contents =
      element Elem { structure_file_element }
    & element MCR { marked_content_reference }
marked_content_reference =
      attribute ref { string }
    & attribute Path { pdf_text_string }

structure_file_element = structure_tree_element_dictionary
structure_tree_element_dictionary =
      attribute Tag { name }
    & attribute Role { name }?
    & attribute Path { pdf_text_string }
    & attribute ref { string }?
    & attribute ID { pdf_byte_string }?    & attribute ID_enc { token }?
    & structure_class
    & attribute Revision { integer }?
    & attribute Title { pdf_text_string }?
    & attribute Lang { pdf_text_string }?
```

```
    & attribute Alt { pdf_text_string }?
    & attribute ActualText { pdf_text_string }?
    & attribute ExpandedName { pdf_text_string }?
    & structure_tree_a

structure_tree_a =
    element Attributes { structure_tree_a_array }
structure_tree_a |=
    element Attribute { attribute_object }
structure_tree_a_array =          attribute_object_or_rev_number*
attribute_object_or_rev_number =
    element Attribute { attribute_object }
attribute_object =
    attribute Owner { name }   & attribute Owner_enc { token }?
  & element UserProperties { array_of_user_property_dictionaries }?
  & attribute Placement { name }?   & attribute Placement_enc { token }?
  & attribute WritingMode { name }?   & attribute WritingMode_enc { token }?
  & attribute BackgroundColor { rgb_color_array }?
  & attribute Color { rgb_color_array }?
  & attribute SpaceBefore { number }?
  & attribute SpaceAfter { number }?
  & attribute StartIndent { number }?
  & attribute EndIndent { number }?
  & attribute TextIndent { number }?
  & attribute TextAlign { name }?   & attribute TextAlign_enc { token }?
  & attribute BlockAlign { name }?   & attribute BlockAlign_enc { token }?
  & attribute IndentAlign { name }?   & attribute IndentAlign_enc { token }?
  & attribute BaselineShift { number }?
  & attribute TextDecorationColor { rgb_color_array }?
  & attribute TextDecorationThickness { number }?
  & attribute TextDecorationType { name }?   & attribute TextDecorationType_enc { token
}?
  & attribute RubyAlign { name }?   & attribute RubyAlign_enc { token }?
  & attribute RubyPosition { name }?   & attribute RubyPosition_enc { token }?
  & attribute GlyphOrientationVerticaly { name }?   & attribute
GlyphOrientationVerticaly_enc { token }?
  &       default_dict_element*
array_of_user_property_dictionaries =
    element UserProperty { user_property_dictionary }*
user_property_dictionary =
    attribute Name { pdf_text_string }
  & attribute FormattedValue { pdf_text_string }?
  & attribute Hidden { boolean }?
user_property_dictionary &= default_dict_element

structure_class =
    element Class {
    attribute Name { attribute _enc {token}? & name } }
```

```
structure_class |=
    element Classes { structure_class_array }
structure_class_array =
    element Class { class_array_element }*
class_array_element =
    attribute Name { name }    & attribute Name_enc { token }?
}
```

FIGURE .3 *Schema for structure information in the Page Structure file*

16.3.4 Marked Content for Structure

The content elements on the page that are referenced by the structure use the marked content mechanism. However, in the case of marked content elements used for logical structure, there is the requirement that they contain an id attribute, so that they can be referenced.

Example 19 Marked Content in SVG Page contents

```
...
<svg:g pdf:Mark="P" xml:id="0">
    <Props/>
    ... SVG content
</svg:g>
```

16.4 Structure Tree Node Naming and Pathnames

To enable the breakdown and reassembly of the structure tree in a predictable and reliable way in the face of page removal and insertion as part of document assembly, each node in the tree is given a name that is unique among its siblings. The node names must be assigned such that the lexical order (details below) of the names matches the left-to-right order of the siblings in the tree. Once the nodes are named, we define the pathname for a node as the multi-segment name whose segments are (in left-to-right order) the name of the root node followed by the name of each node on the path from the root to the node in question. We use the "/" character to separate name segments.

In an ideal PDF document, the structure nodes might have path names as follows:

/Document
/Document/Chapter1
/Document/Chapter1/Paragraph1
/Document/Chapter1/Paragraph2
/Document/Chapter2/Paragraph1
/Document/Chapter2/Paragraph2

In most cases, using the name of the node appended to the path is not sufficient to produce a correct ordering. In the case of a section (Sect) that contained a Title (Title) and a Paragraph (P), sorting lexically would produce an incorrect ordering. Therefore, it is important to disassociate the names of the nodes in the tree with the string used to generate the path. No attempt is made to reconstitute the names of nodes using the information present in the Path string.

16.4.1 Node Name Syntax and Semantics

The name of a node is defined as follows:

NodeName	-> (Series ":")? Segment ("." Segment)* ("-" NodeTag)?
Segment	-> Digit+
Digit	-> "0" | ... | "9"
Series	-> identifier
NodeTag	-> identifier
identifier	-> any graphic Unicode character except "-", "/", ":", and "."

Examples:

1.1.1
12
123.456.76543.345654.3321-HereIsSomeInfo
Ads:1
Ads:2
Ads:1.5
A:3.1-Sect
A:3.2-Sect

The ordering of NodeNames is defined as follows:

1. Nodes are first sorted alphabetically by the Series. If the Series is not present, it is considered to be the null string which sorts before any non-null strings. Sorting "alphabetically" here is defined as comparing characters one at a time from first in the string to last in the string. Characters with greater Unicode character values are considered to come after characters with lesser Unicode values. If one string is a prefix of another, the longer one comes after the

shorter one. If the two strings are equal, comparison continues with the next step.

2. The Segments are then compared by converting each Segment of digits to an integer and comparing those integers. Larger integers are considered greater than smaller ones. Leading zeros are ignored. If corresponding segment numbers are equal, comparison proceeds to the next segment. If only one NodeName has a next segment (because one NodeName is a prefix of the other), the longer NodeName is considered greater.
3. Finally, if two NodeNames are identical up to the trailing NodeTag, then the NodeTags are compared alphabetically in the same manner as the Series.
4. Although they should not appear, syntactically invalid NodeNames should be compared in the same manner as detailed in the previous steps if the invalid part is not required during an otherwise successful comparison. If an invalid segment is encountered it should be considered greater than the corresponding legal segment, regardless of the value of the legal segment.

The segment part of a NodeName may not consist of all zeros. This requirement guarantees that it is always possible to have create a NodeName that comes before a given NodeName. For example, “0.0.0” or “0” are not allowed; “0.0.1” is allowed.

Examples:

1 < 2
1.1 < 2
1.2.1.1 > 1.2.1
0.0.1 < 0.1
9 < 10
2.98 < 2.99
a:1.1 < b:1.1
1.1 < a:1.1
1.23-beta < 1.23-betafish
95.43-alpha < 95.43-beta
1.2.3.4 < 1.2.xyzzy.4 (illegal value)

Rational for Node Naming Syntax

The naming structure is defined to meet several requirements. First, and especially for machine generated node names, a simple integer as the name will often suffice. Thus, the root node would have the name “1”, its first child would have the pathname “1/1”, its second child would have the pathname “1/2” and so on.

This scheme by itself does not work well if editing programs make incremental modifications as insertion of a new structure node in the tree could require significant renumbering of nodes whose names may appear in a large number of locations. To meet this need, the "." segments are added. This allows new nodes to be inserted between, before, or after any existing nodes without renaming the existing nodes. One example of this is the insertion of a new section in a document.

Another likely manipulation of document structure is the merging of orthogonal material such as article content and advertising content in a publication. The structure tree for these two parts of the document would usually remain segregated. Although large gaps in the numbering of the root nodes could be used to achieve this, a more general approach would be to place the two trees effectively in different "name spaces" by giving the root node of one a string prefix of some sort (in this example, perhaps the root node would be named "ads:1"). This guarantees that the node names sort into disjoint groups.

Finally, the suffix allows addition of a human readable name string to help identify structural elements or the addition of a non-numeric key to supplement numeric ordering.

One key aspect of the naming structure is that the additional features do not complicate cases where they are not required.

16.4.2 Node Path Names

The path name to each node consists of joining the node names of adjacent nodes with a "/" separator. A leading "/" is also included to indicate that the path starts at the root of the structure tree. For example, if a page structure file contains the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Structure>
    <Elem Tag="Sect" Role="Sect" Path="/1"/>
    <Elem Tag="heading 1" Role="H1" Path="/1/1"/>
    <MCR Path="/1/1/1" ref="pg.svg#head1"/>
    <Elem Tag="Body Text" Role="P" Path="/1/2"/>
    <MCR Path="/1/2/1" ref="pg.svg#para1"/>
    <Elem Tag="Body Text" Role="P" Path="/1/3"/>
    <MCR Path="/1/3/1" ref="pg.svg#para2"/>
</Structure>
```

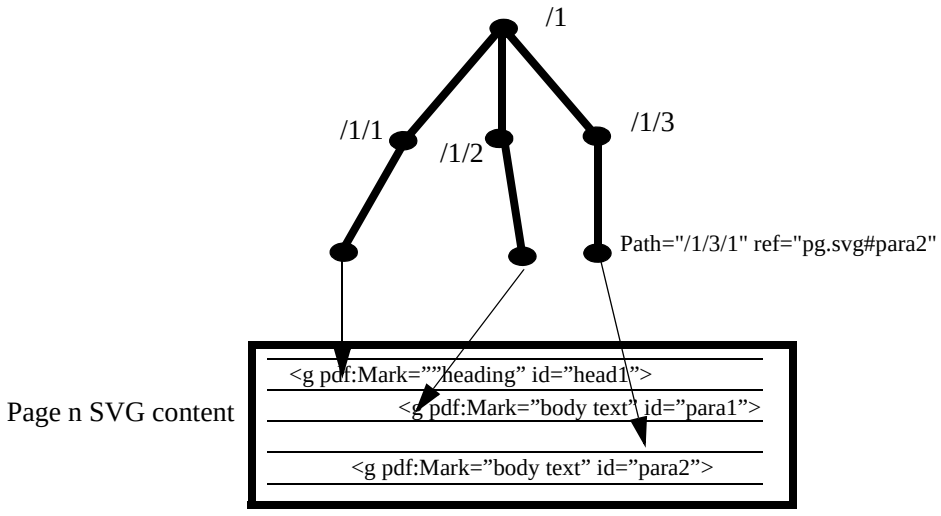
then the structure tree nodes defined are:

TABLE 24

PATH	NODE TAG	ROLE	NOTES
/1	Sect	Sect	Root element
/1/1	heading	H1	
/1/1/1			marked page content reference
/1/2	Body Text	P	
/1/2/1			marked page content reference
/1/3	Body Text	P	
/1/3/1			marked page content reference

and the tree would look like:

FIGURE .3 *Example Structure Tree for a page*



16.4.4 Structure Tree Assembly

The structure tree itself is broken into parts and the parts are associated with individual pages. The node naming scheme enables reassembly of the full tree as needed. The structure tree cache also contains information outlining the full tree structure.

The tree can be reassembled in its entirety or incrementally as needed. To reassemble the tree, the following steps can be used:

1. Locate all of the structure elements (<Elem> tagged XML elements) from all of the pages. If the entire tree is not being assembled, locate only those <Elems> that are being reassembled.
2. Sort all of the elements by Path attribute using the NodeName ordering given above. This will produce a list of nodes in the order they should be created with parents created before children, and earlier siblings created before later siblings.
3. If the entire tree is not being generated, keep track of the pathname of each node so that later additions to the tree can be placed in the proper order.
4. Create the nodes using the sort order and relative positioning.

The structure tree cache (see “Structure Tree Cache” on page 125) contains the full list of structure tree paths sorted which can be used to facilitate the construction process.

16.4.5 Duplicate Information Rules

Each page includes its fragment of the structure tree up to the root. This implies that some structure tree nodes are included on more than one page. Mars files may not contain conflicting information in different structure files for individual pages. It is permissible for some page structure files to contain more structure information than others. Page structure files with an entry for the same pathname may not contain conflicting values for that structure element.

There must be one page structure file that contains the union of all information listed for a given structure element. The structure cache must reference that page structure file.

16.5 Structure Tree Cache

While the breakdown of the structure tree and its distribution among the pages is sufficient for document assembly and content extraction, operations which traverse the document using the structure tree would need to read in all pages in advance to cause recreation of the tree in order to do the traversal. To eliminate this need and its potentially adverse performance impact, the shape of the structure tree is cached and stored in the document cache directory, "/cache/struct.xml".

The cache contains an entry for each structure tree node indicating which page contains information about it. Since more than one page may contain (the same) information about a node, any such page can be used in the cache entry. The cache itself lists all of the structure tree nodes and their path names; this is sufficient information to allow a traversal of the tree. Individual nodes can then be filled in on demand.

Example 20 Sample Structure Cache

```
<?xml version="1.0" encoding="UTF-8"?>
<Cache xmlns="http://ns.adobe.com/pdf/2006"
        Identifier="http://ns.adobe.com/pdf/2006/cache/structure">
    <Query>
        <Files>
            <Pattern Value="/page/*/struct.xml"/>
        </Files>
    </Query>
    <Data>
        <Elem Path="/1" ref="/page/0/struct.xml"/>
        <Elem Path="/1/1" ref="/page/0/struct.xml"/>
        <Elem Path="/1/1/1" ref="/page/0/struct.xml"/>
        <MCR Path="/1/1/1/1" ref="/page/0/struct.xml"/>
        <Elem Path="/1/2" ref="/page/0/struct.xml"/>
        <MCR Path="/1/2/1" ref="/page/0/struct.xml"/>
...
        <Elem Path="/1/10" ref="/page/1/struct.xml"/>
        <MCR Path="/1/10/1" ref="/page/1/struct.xml"/>
        <Elem Path="/1/11" ref="/page/1/struct.xml"/>
        <MCR Path="/1/11/1" ref="/page/1/struct.xml"/>
        <Elem Path="/1/12" ref="/page/1/struct.xml"/>
        <MCR Path="/1/12/1" ref="/page/1/struct.xml"/>
        <Elem Path="/1/13" ref="/page/2/struct.xml"/>
        <Elem Path="/1/14" ref="/page/2/struct.xml"/>
    </Data>
</Cache>
```

See “Structure Cache Schema” on page 130 for more details.

17 Encryption

[Preliminary Section]

18 Signatures

[Preliminary Section]

19 Rights Enablement

[Preliminary Section]

20 Caches

One of the design objectives of Mars is to make it easy to create and manipulate content. Having multiple redundant or cross linked representations of information in the XML can make this quite difficult. The Mars design takes the approach of having the basic representation be “pure and simple” and to then supplement the XML with “cache” data that answer specific queries that would be complex or expensive to run in absence of the cache.

Each cache is a file in the /cache folder in the document package. Caches are either predefined or self-describing. A predefined cache is one that is defined by this specification and whose definition appears in a section below. A self-describing cache is one whose cached data can be recomputed using only information in the cache file itself and the contents of the document package. Self-describing caches provide an extensible cache mechanism. Mars document processors that update documents must update any cache files that are affected by changes they make.

In a self-describing cache, part of the cache file provides a pattern which selects a set of files in the package. A second part of the cache file specifies an xpath, xquery, or limited xslt expression to be applied to the selected files. The final part of the cache file (the data part) contains the XML which is a result of the evaluation of the expression, query, or script.

The cache mechanism is extensible and any number of caches can be present. Since they are self-describing, any Mars producer can create or update all the caches present in a Mars document.

TABLE 25 Mars Required Caches

DESCRIPTION	FILENAME & IDENTIFIER	PATTERN[A]	XPATH/XSLT
Named Destinations. Maps named destination name to file containing destination definitions for the references page.	/cache/names.xml http://ns.adobe.com/pdf/2006/cache/destinations	/page/*/dests.xml Must sort result.	(predefined)
Structure Tree Directory. Maps interior structure tree nodes to a set of pages containing information required for filling in children of that node.	/cache/struct.xml http://ns.adobe.com/pdf/2006/cache/structure	/page/*/struct.xml Must sort result.	(predefined)

a.These patterns assume that the files in the document package are arranged per the recommendations in this specification. If they are arranged differently, the cache pattern must be altered so that the required files are referenced.

The general schema for a cache file is as follows

TABLE 26 *Cache File Schema*

default namespace pdf="http://ns.adobe.com/pdf/2006" grammar { start = cache name = xsd:string cache = element Cache {cache_header, cache_body}	The cache consists of a header and data sections.

TABLE 26 *Cache File Schema*

cache_header =
element Query {
element Files
{ element Pattern { attribute Value { text} }+
}?
& attribute XPath { text }?
& attribute Translate { text }? }
& attribute Identifier { text }?
& element Timestamp {
attribute Type { text }?
attribute Time { text }?
attribute DocumentID { pdf_base64_stirng }?
attribute InstanceID { pdf_base64_string }?
}?
	Files is a pathname with wildcards the select files from which to cache data. There can be one or more patterns. See “File Selection Patterns” on page 128.
	XPath or Translate or both are applied to those files. The resulting nodes are concatenated and placed into the cache Data element. See “XPath and Script Processing” on page 128.
	Identifier is a unique URI name for this type of cache.
	Timestamp is an optional element that allows a test for cache data obsolescence. It is either time-based or document and instance id based.

cache_body = element Data { cache_data } cache_data = ...	The cache_data is specific to the type of cache.

20.1 File Selection Patterns

A file selection pattern, which appears in the Value attribute of the Pattern element is a prototype pathname which may have wildcard characters. The patterns are matched against the fully qualifies names of package files. A package file is matched by a pattern if its fully qualified name, represented as a string, matches a pattern.

A pattern may include zero or more occurrences of the character ‘*’ which matches zero or more other characters. A match occurs if there is any interpretation of what characters a ‘*’ matches that result in all characters in the pattern matching all characters in the fully-qualified name.

20.2 XPath and Script Processing

If present, the value of the XPath attribute is an XPath expression which is evaluated against each XML document selected by the <Files> element. All XML

nodes selected by the XPath expression are gather and made children of a single root <Data> element to make an XML document.

Then, if present, the package file named by the Translate attribute is interpreted as an XSLT script. If there was an XPath attribute present, the input document to the XSLT script is its output. If not present, the XSLT script gets a null document as input. The output of the XSLT script must be an XML document whose root element is <Data>. This element is the value to be used for the cache data.

There are restrictions placed on the XSLT script. These restrictions are TBD.

20.3 Obsolescence Checking

20.4 Cache Error Processing

If a required cache is missing, obsolete, or contains errors, a viewing application should report an error to the user. It may try to recreate or repair the cache and proceed or simply give up and refuse to display the document.

20.5 Named Destination Cache Schema

.

TABLE 27 *Named Destination Cache Schema*

cache_body = element Data { cache_data } cache_data = element Dest {dest_info }* dest_info = attribute Name { name } & attribute Page_ref { text }	For the named destination cache, the data is a sequence of Dest nodes which indicate the name of the named destination and the page to which it refers. The page reference it to the page information filename; it is not the page ordinal.

Named destination cache example

```
<?xml version="1.0" encoding="UTF-8"?>
<Cache xmlns="http://ns.adobe.com/Mars"
        Identifier="http://ns.adobe.com/pdf/2006/cache/destinations">
    <Query>
        <Pattern Value="/page/*/dests.xml"/>
    </Query>
    <Data>
        <Dest Name="EXR55" Page_ref="/pages/p0/dests.xml"/>
        <Dest Name="PieChart" Page_ref="/pages/p0/dests.xml"/>
        <Dest Name="MajorFeatures" Page_ref="/pages/p1/dests.xml"/>
        <Dest Name="DetailedSpec" Page_ref="/pages/p2/dests.xml"/>
    </Data>
</Cache>
```

20.6 Structure Cache Schema

TABLE 28 *Structure Cache Schema*

cache_body = element Data { cache_data } cache_data = structure_file_contents* structure_file_contents = element MCR { marked_content_reference } & element Elem { structure_file_element } marked_content_reference = attribute ref { string } & attribute Path { pdf_text_string } structure_tree_element_dictionary = attribute Path { pdf_text_string } & attribute ref { string }	For the structure cache, the data is a sequence of MCR elements which represent the leaf nodes of the structure tree. For each node, the full pathname is included along with a reference to the page information file for the page which contains the full information for the node. The path names indicate the full list of interior nodes. There can also be Elem elements which represent structure tree leaves that have no content.

21 Foreign XML

21.1 Representing PDFC Extensions

PDF supports extensions defined by third parties. Those extensions are generally ignored by tools other than those that are specifically designed to create and process them.

PDFC extensions consist of extension data that is either located either in

- PDFC dictionaries where the PDF Reference Manual specifically says that application data is to be placed. No other content is defined, or
- any PDFC dictionary but the name of the dictionary key is not one specifically listed in the PDF Reference Manual.

In either case, the Mars schema does not cover the details of extension data.

To represent the extension data accurately in XML, the following format is used:

TABLE 29 Mars Representation of Extension Data from PDFC Documents

LOCATION	DATA TYPE	FORMAT
For data located in a PDFC dictionary *KEY* represents the dictionary key with illegal XML tag characters escaped. See Section D.3.7 “pdf:tag_name Used As An XML Tag” on page 163	null	<*KEY* type=”null”/>
	integer	<*KEY* type=”integer” Value=”123”/>
	number	<*KEY* type=”number” Value=”123.45”/>
	string	<*KEY* type=”string” Value=”here is text”/> <*KEY* type=”string Value=”Here is weird text” Value_enc=”SHIFT-JIS”/>
	boolean	<*KEY* type=”boolean” Value=”true”/> <*KEY* type=”boolean” Value=”false”/>
	name	<*KEY* type=”name” Value=”a_name”/> <*KEY* type=”name” Value=”a_name” Value_enc=”BASE64”/>
	dictionary	<*KEY* type=”dictionary”> nested dictionary values </*KEY*>
	array	<*KEY* type=”array”> nested array elements </*KEY*>
	stream	<*KEY* type=”stream” src=”stream file within package”> nested dictionary values </*KEY*>
For data located in a PDFC array	null	<Null/>
	integer	<Int Value=”123”/>
	number	<Number Value=”123.45”/>
	string	<String Value=”this is text”/> <String Value=”r7d3h4hfcD==” Value_enc=”BASE64”/>
	boolean	<Bool Value=”false”/>
	name	<Name Value=”A Name”/> <Name Value=”encoded name” Value_enc=”BASE64”/>
	dictionary	<Dict> nested dictionary content </Dict>
	array	<Array> nested array content </Array>
	stream	<Stream src=”file within package containing stream”> nested stream dictionary content </Stream>

This representation provides a means for representing any values in a PDFC document that are not recognized as a defined part of the PDF language, and a means for a Mars creator to define XML that will be mapped into a known PDFC structure.

Example 21 Mars representation of COS extensions

```
<FICL3aEnfocus type="dictionary">
    <PitStop type="dictionary">
        <CC type="dictionary"/>
    </PitStop>
</FICL3aEnfocus>

<ParmDictionary>
    <K Value="1" type="integer"/>
    <Columns Value="8" type="integer"/>
</ParmDictionary>
```

21.2 Where PDFC Extension Data Can Go

There are explicit locations where extension data can be located. These are defined in the PDF Reference Manual and appear in the Mars schema as “default_dictionary”. Locations include:

TABLE 30 Locations Where PDFC Extension Data is Explicitly Allowed

SCHEMA RULE	TAG	NOTES
property_list_dictionary	Custom	
custom_annotation_dictionary	Custom	
application_data_dictionary	Private	
signature_dictionary	TBD	not part of initial Mars release
common_stream_dictionary		directly in stream dictionary
common_stream_dictionary	DecodeParms	for stream filters
user_property_dictionary	UserProperty	In PDFC, the custom content is under the ‘V’ key.
attribute_object		Tagged PDF attributes

In addition, extension data is supported in the following locations:

TABLE 31 Additional Locations Where PDFC Extension Data Is Allowed

SCHEMA RULE	NOTES
catalog_dictionary	
page_dictionary	
font_dictionary	
viewer_preferences_dictionary	
custom_annotation_dictionary	
common_annotation_dictionary	
custom_action_dictionary	
oc_usage_dictionary	oc_creatorinfo_dictionary
halftone_dictionary	for custom halftone types

21.3 XML Extensions in Mars

21.3.1 Behavior of Processors When Extensions Are Encountered

Mars XML files can, in general, contain non-pdf namespace extensions. SVG page content files also can contain non-pdf and non-svg namespace extensions. When such unrecognized extensions are encountered by Mars processors, they should be ignored and normal processing of the Mars document should proceed as though the extensions were not present.

21.3.2 Preservation of Extensions By Processors

Although extensions can appear on most elements, it is not reasonable to expect all Mars document processors to preserve all extension data when documents are

modified and rewritten. To support workflows involving extension data, there is a minimum preservation requirement.

1. Any extension data represented in the pdf namespace using conventions defined in “Representing PDFC Extensions” on page 131 must always be preserved by Mars processors.
2. Any non-pdf namespace XML extension data (elements and attributes) must be preserved if placed in an element as defined in Table 30 on page 133 and Table 31 on page 134.
3. If a change to the document results in the deletion or replacement of an element containing extensions data of either kind, then the extension data should be deleted and not appear as part of the new document content.
4. Preservation of extension data at other locations is implementation-defined. It is not precluded.

Appendix A Schema Guide

The schemata for Mars are listed in the following table. Some of these schemata appeared earlier in the document. The current, most definitive schemata are in the files referenced below.

Note: *To use the hyperlinks in this table, place the schema files in the same folder as this document.*

TABLE 32 Mars Schema Guide

SCHEMA	FILE	SECTION	NOTES
Backbone	backbone.rnc	5 pg 28	
Bookmark	bookmarks.rnc	5.4 pg 33	
Article Threads	articles.rnc	5.11 pg 36	
Application Data	appdata.rnc	5.8 pg 35	
Default Annotation Appearances	appearances.rnc	5.12 pg 37	
JavaScripts	javascripts.rnc	5.13 pg 37	
Web Ids	web_ids.rnc		
Web URLs	web_urls.rnc		
Embedded Files	embedded_files.rnc	5.14 pg 37	
Page Information	pageinfo.rnc	6.1 pg 38	
Page Structure	pagestruct.rnc	16.3 pg 116	
Font Descriptor	fontdesc.rnc	10.3 pg 79	
Markup Annotations	markup_annot.rnc	13 pg 96	
Content Annotations	content_annot.rnc	13 pg 96	
Named Destinations	pagenameddest.rnc	6.4 pg 41	
Named Destination Cache	See section	20.5 pg 129	
Structure Tree Cache	See section	20.6 pg 130	

TABLE 33 RelaxNG References

TITLE	REFERENCE
Oasis RelaxNG Specification	http://www.relaxng.org/spec-20011203.html
Oasis RelaxNG Compact Syntax	http://www.oasis-open.org/committees/relax-ng/compact-20021121.html
RelaxNG Compact Syntax Tutorial	http://relaxng.org/compact-tutorial-20030326.html
Eric van der Vlist. Book for O'Reilly	http://books.xmlschemata.org/relaxng/
XML Schema Datatypes	http://www.w3.org/TR/xml schema-2/

Appendix B Conversion Between PDF and Mars

It is possible to convert between the Mars SVG representation and the PDF page content representation of a page.

Mars with SVG page content can be read and displayed in Adobe Reader and converted to PDFC graphic operators using Acrobat Professional.

B.1 Correspondence to PDF Document Content

This section gives a brief overview of how various features of PDF are handled in Mars. It is most useful to those familiar with PDF.

Catalog Dictionary

The information in the PDF document catalog appears as top-level elements in the document backbone. The structure tree and named destinations are handled in significantly different ways and do not appear in the same organization in the catalog dictionary. Bookmarks, web capture, and application information are moved to separate files to avoid open-time delays when this information is not immediately required.

Page Contents

In the document backbone, the page tree is flattened as a series of XML <Page> elements. Each of these points to page information which is stored in separate files. The graphic page content itself is represented using SVG. Some structure tree and marked content information is included on the page interspersed with the SVG itself. Additional structure information, named destination information, resources, and annotations are stored in separate files associated with the page to which they apply.

Fonts

Each font consists of a font descriptor file and the font file itself (if the font is embedded). The font descriptors are represented in XML and can be shared

across pages in a Mars document. The fonts themselves are represented in OpenType format or as an SVG font.

Annotations

Annotations are placed in separate files, grouped by page. For each page, there can be two annotation files, one containing markup annotations such as sticky-notes and text comments, and another containing content-oriented annotations such as form field widgets and hyperlinks.

Bookmarks

The bookmarks (outline items in PDF) are represented in XML in a separate package file that is referenced from the document backbone file.

Metadata

Document level metadata is stored in a separate file in the META-INF directory and contains the XMP representation of the metadata. Metadata associated with other package objects such as pages, fonts, or images, appear in separate files related to the files that they describe.

The Info dictionary information is not represented explicitly in a Mars document; only the XMP metadata is present.

Object-level metadata which describes page-level graphic objects is represented using marked content XML markup within the SVG page contents file.

Logical Structure and Tagged PDF

In PDF, document structure information is represented partially in a structure tree and partially in page content. In Mars, the XML structure is intended to be easier to create and manipulate. Structure information for a particular page is represented as markup that is part of that page, making it easier to create pages and move pages between files.

The role map and class map are stored as part of the document backbone. In addition, roles and classes used on a given page are also listed on that page.

The structure tree is broken down as follows: leaf nodes are represented as part of markup corresponding to marked content containers nested within the SVG page content. Information about interior structure nodes that are parents of the content on a page is stored in a separate file associated with that page. Finally, information indicating which pages have the structure information for which interior nodes is stored in a cache file. For more detail, see section 16, “Marked Content & Logical Structure”.

Web Capture

Some web capture information is stored in the document backbone. This information generally provides the context in which the PDF was created from web pages and allows the PDF to be updated from the same web pages. The URL and ID maps that are in the Name dictionary in PDF, are stored in separate package files that are referenced from the document backbone.

Optional Content

Optional content groups, configurations, and so on are stored in the document backbone. Optional content information that is part of the page content is stored as markup on the SVG page.

Multimedia

Media selection, rendition information, player information, and so on are stored in the document backbone. Movie annotations are part of the content annotations stored with its associated page. Sound annotations are markup annotations also stored with its associated page. The annotations themselves refer to the media files which are within the document package.

File Attachments

Embedded file information is stored within the document backbone. File attachment annotations are considered markup annotations stored with other annotations associated with a page. The attached file itself is stored as a separate file within the document package.

Pages

The basic list of pages is stored within the document backbone. The information about the page itself is broken into up to 8 or more files: the page information file, the page content file, the page structure file, the named destinations file, the content annotations file, the markup annotations file, page-level metadata (xmp) file, the page resources file, and any images, fonts, font descriptors, functions, shaders, patterns, color profiles, and others. In addition, the page content itself can be broken into multiple files using the PDF "form xobject" mechanism (PDFC) and the SVG symbol mechanism (Mars).

Page Resources

Page resources as they appear in PDFC (including, fonts and images) are represented by markup that is part of the SVG page on which they appear. In some cases, SVG content will refer to other files using a URI. These external resources can be shared by multiple pages. Other PDFC page resources are converted entirely to SVG and represented only in the SVG page contents file.

Color Spaces

Some PDFC color spaces are converted to ICC Color profiles. Others have a Mars-specific representation. The color space definitions appear on the SVG page which references them or in a shared resource file. The ICC profiles are often separate package files referred to by the SVG page resource definition.

Named Destinations

PDFC stores named destinations in a central name tree or array. In Mars, the named destinations are broken down, grouped, and stored with the page to which they refer. A separate cache file maps the names to the page file where the details of the destination are stored.

Streams

Each object that is represented as a stream in PDFC is placed in a separate file. This includes page contents, fonts, images, ICC color profiles, user file attachments, form data, and other objects.

Form Fields

Form fields in PDF consist of three parts: some base information in the Catalog dictionary (the Acroform dictionary), form fields (each of which is represented by a Field Dictionary), and form field Widgets (each of which is represented by a Widget Annotation dictionary). This representation carries over into Mars. Base information about the form is in the AcroForm element within the document backbone. In addition to general information about the form, this element includes elements representing each form field. The form field widgets, representing the visible manifestations of the fields that are displayed on the page(s) of the document, are stored in the content.can file associated with each page.

In PDFC, the widget annotation dictionary and field dictionary can be shared if there is a 1-1 correspondence between them. Such sharing is not done in Mars.

The field elements refer to the widget annotation elements using an uri reference scheme. This crosses files as the fields are in the backbone and the widgets are in annotation files referenced by the pages on which they appear.

The widgets are likely to refer to appearance streams which are SVG content files that describe how to render them. These SVG files are separate package objects that should be in the page directory (if they are used only on that page) or in the xobjects directory (if they are used on multiple pages).

XFA Forms

XFA forms, in addition to the information present for a non-XFA form, have one or more XFA files. In PDFC, these are either in a single, XDP-format file (deprecated) or split into multiple PDF streams where each stream corresponds to an XDP packet. In Mars, the individual XDP packets appear as separate package files referenced by the backbone.

B.2 PDF Features Not Supported in Mars

B.2.1 PDF Graphic Features Not Supported in Mars

Some features of PDF are not be supported in Mars's SVG format.

TABLE 34 PDF page content features not supported or partially supported in Mars

NAME	DESCRIPTION
bx / ex operator	In PDF, compatibility sections wrapped in bx and ex are content which can be ignored if not recognized. Such a construct is not supported in Mars (although the SVG switch construct is similar). When converting PDFC to Mars, the bx/ex operators are removed; the enclosed content is retained.
CID Fonts	CID Fonts are not supported in Mars. When converting PDFC to Mars, CID fonts are converted to OpenType and do not round-trip back to CID fonts. The representation of characters on the pages may be changed as part of this conversion.
Type 3 Fonts	Type 3 fonts are not supported in Mars. They can be converted to SVG fonts which are supported.
Pass-Through PostScript	PostScript XObjects are not supported. When Acrobat converts from PDFC to Mars, presence of pass-through PostScript will result in user warnings and removal of the PostScript content.

B.2.2 Other PDF Features Not Supported In Mars

TABLE 35 Other PDF features not supported or partially supported in Mars

NAME	DESCRIPTION
Stream Valued Structure Attributes	Per PDF Reverence Manual 1.6 Table 10.10 Key A, structure attributes can ultimately be a dictionary or stream. In Mars, stream valued attributes are not supported.
Attribute Revision Numbers	Revision numbers for attributes are not kept with structure tree elements or with attribute or class references.
Name key in Soft Mask	Name for soft mask is deprecated.
Structured Bookmarks	Ability for bookmarks to point into the structure tree.

B.2.3 PDF Features Not Round-Tripped in Mars

CID Fonts

B.2.4 Mars SVG Features Not Round-Tripped

SVG non-primitive content (e.g., circle, square)

SVG Extension content.

B.3 Notes on Converting Between PDF and Mars

B.3.1 Storing Foreign XML In PDF

This section explains the conversion process for data that is not found in the pdf namespace when a Mars document is converted to PDF.

XML elements and attributes outside the pdf namespace are processed in a manner that results in no loss of information when a PDFC document is converted to Mars and back again to PDFC if they are placed in locations identified for preservation of such data.

User-defined XML elements and attributes in namespace other than the Mars namespace that appear in Mars XML content (other than page contents) are handled as follows:

- A “UserXML” key is added to the PDFC dictionary that corresponds to the Mars element
- For elements appearing within known Mars elements: a dictionary entry is created in the dictionary whose key is “XMLExtension” and whose value is a string consisting of the concatenated element values. An element value is represented in PDFC as a string that is the canonicalized XML element, including its children.
- For attributes appearing within known Mars elements: a dictionary entry is created in the UserXML dictionary whose key is “XMLExtensionAttrs” and whose value is a string consisting of the concatenated attribute names and values.

- Comments and processing instructions are ignored and not preserved. Text data in known Mars elements that are not defined to include text is also ignored and not preserved.

Examples:

```
<PDF Case1a="Hello" Case1b="Hi">
<Pages>... </Pages>
<Case1 foo="23"/>
<Case2>24</Case2>
<Case3 foo="25">26</Case3>
<Case4>
27
<Orion>28</Orion>
<Omicron bar="29"/>
</Case4>
<Case5>98</Case5>
<Case5>99</Case5>
</PDF>
```

turns into

```
<< /* other catalog dictionary stuff *?
/XMLExtensionAttrs  (Case1a="Hello" Case1b="Hi")
/XMLExtension (<Case1 foo="23"/>
<Case2>24</Case2>
<Case3 foo="25">26</Case3>
<Case4>
27
<Orion>28</Orion>
<Omicron bar="29"/>
</Case4>
<Case5>98</Case5>
<Case5>99</Case5>)
>>
```

XML extensions in SVG page contents are discarded on input and neither represented in the PDF content stream nor preserved should the document be resaved after modification. (Actual behavior is that the extension data is removed only if the page is actually modified in some way. Otherwise, the page is resaved unchanged.)

B.3.2 Restrictions on PDF Extensions

Table 31 on page 134 lists places where PDF extensions can appear in Mars XML. The reason for this explicit enumeration is that we can't reliably round-trip PDFC extensions that are stuck in arbitrary places. There are two problems: (1) Mars sometimes represents PDFC data structures in some completely differ-

ent way, with no promise to round-trip back to the original PDFC. (2) In the Mars representation you don't know which things were originally dictionaries (which can store extensions in PDFC) versus arrays (which can't).

B.3.3 SVG and Graphics

Determination of presence of transparency on a PDF page: Currently, transparency in PDF is deemed to be present if, in the resource dictionary for a page, a) an SMask if found, b) a non-Normal blend mode is found, or c) /CA or /ca have a value of less than 1. We also look at image resources and see if they have a /SMaskInData with a value greater than 1 which can only happen for JPEG20000 encoded images.

B.3.4 Coordinate Differences

Mars follows the SVG style of coordinates with the origin being in the upper left corner of a page image. This is different than the PDF definition of the origin being in the lower left corner and changes the notion of "up" on the page from going to higher values of y coordinates to going to lower values of y coordinates. Conversion between the coordinate system depends on knowing the height of the page or area being converted. This height, H, is defined as:

- For pages, H is UserUnit * Ymax of the media box. In PDFC documents, UserUnit and MediaBox are in the page's page dictionary. Ymax is at index 3 of the MediaBox array. In Mars documents, UserUnit is the (optional, default 1.0) UserUnit attribute of the page's Page element, and YMax is the y2 attribute on the page's Page element.
- For other types of content, H is Ymax of the bounding box of the element.

Conversion is handled as follows:

TABLE 36 Conversion of coordinates and matrices

GIVEN...	CONVERT TO	VALUE IS
PDF coordinate (x, y) and page height H	SVG coordinate	(x, H-y)
SVG coordinate (x, y) and page height H	PDF coordinate	(x, H-y)

TABLE 36 Conversion of coordinates and matrices

GIVEN...	CONVERT TO	VALUE IS
PDF matrix (a, b, c, d, x, y) or in matrix form: $\begin{bmatrix} a & c & x \\ b & d & y \\ 0 & 0 & 1 \end{bmatrix}$	SVG-style matrix	$\begin{bmatrix} a & -c & x + cH \\ -b & d & H(1-d) - y) \\ 0 & 0 & 1 \end{bmatrix}$
SVG-style matrix (a b c d x y) $\begin{bmatrix} a & c & x \\ b & d & y \\ 0 & 0 & 1 \end{bmatrix}$	PDF-style matrix	$\begin{bmatrix} a & -c & x - cH \\ -b & d & -y + H(1-d)) \\ 0 & 0 & 1 \end{bmatrix}$

B.3.5 PDF Info Dictionary

There is no Mars analog of the PDF Info dictionary. Mars uses only a document level XMP file for storing metadata. When converting from PDF to Mars, the Info dictionary is discarded. When converting from Mars to PDF, the Info dictionary is created from the XMP using the contents of the pdf metadata schema.

B.4 Representation Differences Between PDF and Mars

B.4.1 Named Destinations

In PDFC, named destinations are stored centrally either in a dictionary or in a name tree.

B.4.2 Annotations

In PDFC annotations for a page are stored in the page dictionary for that page. PDFC does not distinguish between annotation types and treats them all as a group.

B.4.3 Fonts Descriptors

(Note: the Mars font descriptor files correspond to the information in PDFC Type 0, Type 1, Type 3, and TrueType font dictionaries.)

B.4.4 Representation of PDF Fonts

PDF supports a number of different font representations. Those representations and how they are handled in Mars is show in the following table.

TABLE 37 Mars Support of PDF Fonts

PDF FONT	NOTES	MARS REPRESENTATION
Type 1	Single Byte Encoding	OpenType
TrueType	Single Byte Encoding	OpenType
Type 3	Single Byte Encoding, can have multi-color glyphs	SVG Font
Type 0 + CIDFontType0	Multi-byte Encoding	OpenType using CFF style glyphs. Content converted to Unicode or glyph ids specific to font. CID fonts are not supported in Mars.
Type 0 + CIDFontType2	Multi-byte Encoding, TrueType-style glyphs	OpenType. Content converted to Unicode or glyph ids specific to font. CID fonts are not supported in Mars.
OpenType		OpenType
Multi-Master	Only 1 instance is embedded so this is really the same as a type 1 font.	

TABLE 37 Mars Support of PDF Fonts

PDF FONT	NOTES	MARS REPRESENTATION
TrueType Collection	Not supported	None

Fonts that use the TrueType or OpenType format do not need to be converted when translating between Mars and PDF. These fonts can be used as is.

B.4.5 Character Maps

To support mapping Unicode characters to and from font glyphs, a Unicode Character map (CMap) may be present and associated with a font. Unicode CMaps are described in the PDFR 1.6 section 5.9.2 and Adobe Technical Notes referenced there.

Unicode CMaps in PDF fonts are added to the font itself if required. Thus, in Mars, the font is the primary and sole container of Unicode mapping information for the font.

When converting from Mars to PDF, the unicode map may be recreated and added to the PDF font dictionary.

Appendix C Recommended Conventions and Practices

1. Use relative URIs for file references. For example, a page should reference “im123.jpg” rather than “/page/23/im123.jpg”. This makes content more modular and easier to reuse. It also tends to reduce file size.
2. Organize page content in reading order to enhance accessibility. This is a requirement for tagged PDF.
3. If there is a desire to have a page svg file displayed in a stand-alone SVG viewer, then a viewport should be established on the svg root element by adding a viewBox attribute with values that define a box corresponding to the crop box and/or media box. Although viewBox is not a supported Mars attribute, this will cause the standalone SVG viewer to apply the same crop to the graphic content as would a Mars viewer. Note that the CropBox is a value “x1 y1 x2 y2” and viewBox is a value “x y width height”.

Appendix D Character Encoding

D.1 Introduction

Many strings in PDF do not have a specific character encoding specified. In XML, all characters have a specific encoding. This section covers how these unencoded strings are moved between PDF and XML.

Encoding of characters is a complex issue. The Unicode standard has defined standard codes for (more or less) all characters. Unicode code-points are described using the form "U+hex character code", e.g., U+0060. Unicode defines roughly the range U+0000 to U+10FFFF (with a few reserved blocks). Unicode characters are represented using an "encoding form" such as UTF-8 or UTF-16 which map Unicode code points to unique code unit sequences. Then, a Unicode encoding scheme such as UTF-16BE or UTF-8, defines a byte serialization for a Unicode encoding form, including the specification of the handling of a byte order mark (BOM), if allowed.

TABLE 38 Definitions and References related to encoding

ENCODING	REFERENCE
Unicode	The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined by: *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1 (http://www.unicode.org/versions/Unicode4.0.1) and by *Unicode 4.1.0* (http://www.unicode.org/versions/Unicode4.1.0)
UTF-8	Introduction and informal description: http://en.wikipedia.org/wiki/UTF-8 Formal Definition: http://www.unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404 http://www.unicode.org/versions/Unicode4.0.0/ch03.pdf#G28070

TABLE 38 Definitions and References related to encoding

ENCODING	REFERENCE
UTF-16	Introduction and informal description: http://en.wikipedia.org/wiki/UTF-16 Formal Definition: http://www.unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404 http://www.unicode.org/versions/Unicode4.0.0/ch03.pdf#G28070
PDFDocEncoding	Also, Appendix D of the PDF 1.6 reference manual
ISO-8859-1 (Latin-1)	Introduction and informal description: http://en.wikipedia.org/wiki/Iso-8859-1 Formal Definition: "ISO/IEC 8859-1:1998 Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1" published by ISO (not available online).
Base 64	RFC 3548 (http://www.ietf.org/rfc/rfc3548.txt)

D.2 Description

There are 4 places that text strings occur in PDF documents. They are

1. objects of PDF type **string**. There is no PDF-wide characterization of what the bytes of a string may mean. They can be characters in any number of encodings, or binary data. Starting in PDF 1.7, these are called **byte strings**.
2. objects of PDF type **text string**. Text strings are defined to be character strings of PDFDocEncoding or the bytes 0xFE followed by 0xFF followed by a UTF-16BE encoded string.
3. objects of PDF type **name**. There is no PDF-wide characterization of what the bytes of a name may mean. They are usually ASCII characters, but can be arbitrary binary data, or UTF-8 encoded characters.
4. objects of PDF type **stream**

Each use of a string, text string, or name in PDF is labelled using a particular type in the Mars schema. The interpretation of the corresponding element or attribute in the Mars XML is based on this type.

TABLE 39 Definition of String-Related Schema Types

STRING TYPE FROM SCHEMA	DEFINITION
pdf:text_string	PDF file encoding is known and is either PDFDocEncoding or 0xFE, 0xFF followed by UTF-16BE. Used when the object is defined as a text string in PDF.
pdf:byte_string	Encoding unknown in PDF file. Bytes usually represent characters, but must be treated as binary data. The conversion process tries to interpret the PDF string as characters and represent them in an intelligent way in the XML. Used in general for PDF strings and names.
pdf:base64_string	String data represents binary data which is encoded in base64 in the XML Used when it is known that a PDF string or name is a binary object.
pdf:doc_encoded_string	String represents characters encoded in PDFDocEncoding, 1 byte per character. See PDF Reference manual 1.6 Appendix D. Used when it is known that a PDF string or name is limited to characters of this encoding.
pdf:text_name	A name value that is composed of UTF-8 encoded text characters.
pdf:name	A name value that may contain arbitrary binary bytes (except nulls)
pdf:tag_name	A name value that is used as an XML tag.

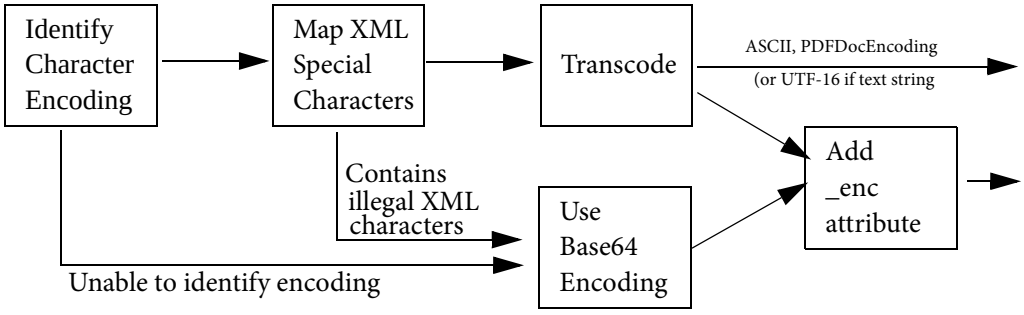
For each of the schema types, there are three questions to answer about processing. These questions are answered in the sections covering each of the schema types, below.

1. What should be done when converting from PDF to Mars?
2. What should be done when converting from Mars to PDF?
3. How should the element or attribute value be processed and interpreted when manipulating the Mars format? The conversion and processing described applies to element content as well as attribute values in the XML.

Note that in the XML file, the characters are always encoded based on the encoding specified in the XML file itself. The schema types below for strings indicate how to transcode the Unicode characters in the XML to and from bytes in the PDF document.

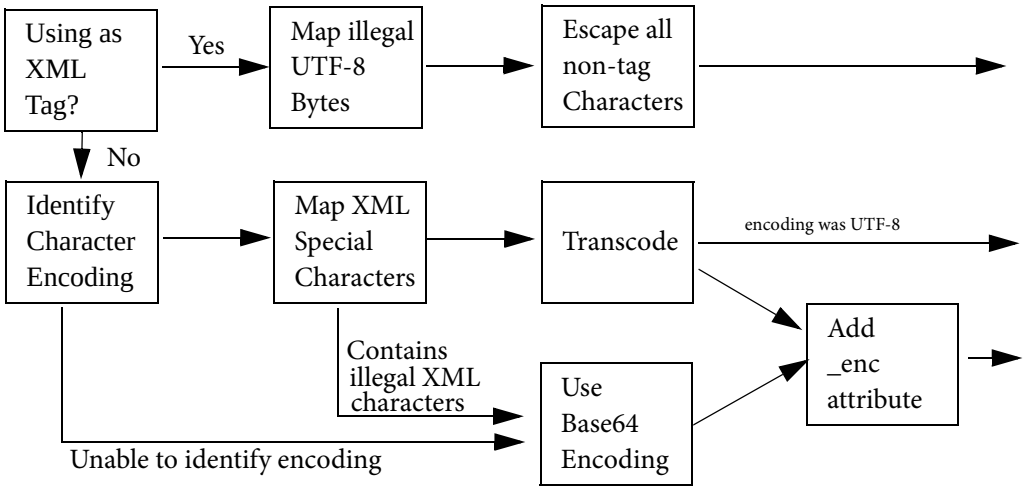
D.3 Overview of the string conversion process

The process for converting strings from PDF format to Mars format is:



1. Try to identify the encoding of the PDF string or name data. Details below.
2. Based on the presumed encoding, map the bytes to Unicode characters.
3. If the string contains characters that are illegal in XML, or would result in data loss, then switch to BASE64 encoding. Specifically,
   - if generating XML 1.0 and the string includes characters in the range U+0000 to U+001F (excluding whitespace characters U+0009, U+000A, and U+000D),
   - if generating XML 1.1 and the string includes character U+0000, or
   - if the string is to be an attribute value and contains significant leading or trailing space (U+0020) characters or consecutive internal space (U+0020) characters (because whitespace normalization on processing the XML will cause the spaces to be lost).
4. Some characters that appear in strings may need to be mapped to an alternate representation to appear legally in XML, for example, you may need to replace "<" with "<". (Some APIs, for example Java™ XML serialization, handle changing the representation of these characters.) Replace special XML characters with an alternate representation such as the corresponding character entities. The characters that need to be replaced and the alternate representation used are based on the XML specification. The information presented here is advisory, not definitive.

The process for converting names from PDF format to Mars format is:



1. If name is to be used as an XML tag, escape all characters that cannot be used in an XML tag. Finished for this case.

1. Try to identify the encoding of the PDF name data. Details below.
2. Based on the presumed encoding, map the bytes to Unicode characters.
3. If the name contains characters that are illegal in XML, or would result in data loss, then switch to BASE64 encoding. Specifically,
   - if generating XML 1.0 and the name includes characters in the range U+0000 to U+001F (excluding whitespace characters U+0009, U+000A, and U+000D),
   - if generating XML 1.1 and the string includes character U+0000, or
   - if the name is to be an attribute value and contains significant leading or trailing space (U+0020) characters or consecutive internal space (U+0020) characters (because whitespace normalization on processing the XML will cause the spaces to be lost).
4. Some characters that appear in names may need to be mapped to an alternate representation to appear legally in XML, for example, you may need to replace "<" with "<". (Some APIs, for example Java XML serialization, handle changing the representation of these characters.) Replace special XML

characters with an alternate representation such as the corresponding character entities. The characters that need to be replaced and the alternate representation used are based on the XML specification. The information presented here is advisory, not definitive.

5. Emit a special attribute indicating the encoding that was identified in step 1 if that encoding is other than PDFDocEncoding and not determinable from the schema type.

6. Emit a special attribute indicating the encoding that was identified in step 1 if that encoding is other than UTF-8 and not determinable from the schema type.

TABLE 40 Special XML characters requiring alternate representation and possible alternative representations

CHARACTER	ENTITY	NOTES
<	<	Must be escaped everywhere
&	&	Must be escaped everywhere
“ ‘	" '	Must be escaped in attribute values (unless single quotes enclose the string with double quotes, or vice versa).
Whitespace: CR (0x0D), LF (0x0A), TAB (0x09), and	
 	Only in attribute values, whitespace characters must be represented as character references if they are significant and would be removed by XML whitespace normalization for attributes.

The string conversion process from Mars to PDF format is:

1. Look for a special attribute indicating the target encoding to be used in the PDF file.

2. Replace character entities with actual characters. See Table 40 on page 156. This is typically done automatically by a conforming XML parser.

3. Transcode the Unicode characters to bytes to place in the PDF file using PDFDocEncoding, an explicit target encoding from step 1 (if present), or information from the schema type.

The overall result of these transformations is that any sequence of bytes can be sent from PDF to XML and back resulting in the same binary value in the PDF, or from XML to PDF and back resulting in the same Unicode characters in the XML.

D.3.1 pdf:text_string

PDFC to Mars

Within a PDFC document, strings with this type have a known encoding. The encoding is either **PDFDocEncoding** (defined in PDF 1.6 Reference manual, Appendix D), or the bytes 0xFE followed by 0xFF followed by a **UTF-16BE** string. Details of this are in the PDF 1.6 Reference manual section 3.8.1, pages 131-132.

When generating XML from PDF, text strings within the PDF are examined for the 0xFE 0xFF prefix bytes. If present, the remainder of the string is interpreted as **UTF-16BE**; if absent, the string is interpreted as **PDFDocEncoding**. Characters should then be transcoded to the appropriate encoding for the generated XML, following the general steps presented in "Overview of the string conversion process" on page 154.

The special encoding attribute is not generated for strings of this schema type.

Mars To PDFC

When reading XML and generating PDFC, the XML characters should be transcoded into either **PDFDocEncoding** or **UTF-16**. The target encoding used in the PDF file can be chosen by the implementation but must be lossless (that is, **PDFDocEncoding** cannot be used if any character is present that cannot be represented using **PDFDocEncoding**. If **UTF-16** is chosen, the PDF string must begin with 0xFE followed by 0xFF followed by the string represented using **UTF-16BE** encoding.

D.3.2 pdf:byte_string

Bytes in strings of this type are not in a known encoding. Usually, such strings are single byte character data, but this may not be the case. The objective in processing these strings is to make them as useful in XML as possible.

PDFC to Mars

The encoding of the pdf:byte_string object is determined based on the following algorithm.

Algorithm 2 ***Identifying the pdf:byte_string encoding***

1. If the string contains only bytes with values Tab 0x09, LF 0x0A, CR 0x0D, and 0x20-0x7F, then the encoding is **ASCII**.
2. if the string begins with the bytes 0xFE 0xFF and contains only legal byte sequences for UTF-16BE, then the encoding is **UTF-16**
3. if the string begins with the bytes 0xEF 0xBB 0xBF and contains only legal byte sequences for UTF-8, then the encoding is **UTF-8**.
4. if an implementation has additional ways to recognize the encoding, they can be applied at this point. If the encoding is positively recognized, that encoding is the result.
5. if none of the above apply, and the string contains only bytes defined by the **PDFDocEncoding**, then the encoding is **PDFDocEncoding**.
6. if none of the above apply, the string is considered binary and represented in a base 64 text representation of the bytes. The encoding is **BASE64**. Special case: if the bytes are to be used as an attribute value and the string contains leading spaces, trailing spaces, or consecutive internal spaces, then the **BASE64** encoding must be used.

If the encoding determined by the above is other than **ASCII** or **PDFDocEncoding**, then a special attribute naming the encoding must be emitted. If the value is going into an XML element, the attribute name is "_enc". If the value is going into an XML attribute, the attribute name containing the encoding value is the attribute name being generated for the string with "_enc" appended.

The value of the _enc attribute must be "BASE64" or should conform to the character set encoding name guidelines given as part of the XML 1.0 specification section 4.3.3 (namely that it be one of "UTF-8", "UTF-16", "ISO-10646-UCS-2", "ISO-10646-UCS-4", "ISO-8859-1", "ISO-8859-2", ... "ISO-8859-n" (where n is the part number), "ISO-2022-JP", "Shift_JIS", "EUC-JP" or that it be a registered name from the Internet Assigned Numbers Authority [See http://www.iana.org/assignments/character-sets.]). Note that this is not the encoding of the value of the Name attribute (which is specified as part of the encoding declaration in the XML). Rather it is the encoding that was used or that should be

used when the file name is converted from or to the byte string stored within the PDF file itself.

Values other than **ASCII**, **PDFDocEncoding**, **UTF-8**, and **UTF-16** should not be used with the expectation that they will be recognized and processed by all tools and implementations.

Example: If the encoding for a pdf:byte_string is identified as UTF-16 and the string itself was to be the value of an element with tag <Value>, then the effect of these rules is to emit an element with tag <Value _enc="UTF-16">.

Another example: If an attribute Filename is being created from a pdf:byte_string whose encoding is determined to be UTF-8, then a second attribute Filename_enc="UTF-8" should be generated.

Mars to PDFC

The XML string is mapped to the PDFC string as outlined in "Overview of the string conversion process" on page 154. For the transcoding step, the containing XML element is examined for presence of a "_enc" attribute or a "name_enc" attribute if the string value itself is coming from attribute "name". If present, then this is the target encoding for the string when it is stored in the PDF file. If absent, **PDFDocEncoding** (which covers **ASCII** as a subset) is the target encoding for the Unicode string.

If the encoding was UTF-16, the PDFC string will be prefixed with the bytes 0xFE 0xFF.

If the encoding was UTF-8, the PDFC string will be prefixed with the bytes 0xEF 0xBB 0xBF.

Processing Byte Strings in XML

Generators and processors of XML fragments that came from pdf:byte_strings need to take care not to introduce Unicode characters that cannot be sent back to PDF if **PDFDocEncoding** is in use. In addition, processors may need to be aware that they are working with data where the encoding has been mis-identified. The string as a whole will round-trip back to PDF correctly even if the encoding is mis-identified, but may not be manipulable by XML tools in between. Processors also need to be prepared to receive **BASE64** encoded binary data if that is the encoding indicated in the _enc attribute.

Creators of content can control the transcoding of Unicode character values by adding the "_enc" attribute. This is only possible for items of schema type pdf:byte_string, however.

D.3.3 pdf:base64_string

A string in PDFC known to be binary data is represented using a base64 encoding within XML. The string must be converted to base64 when extracted from PDF and placed in XML and converted back to binary when creating PDFC from XML information.

Each byte represents 6 bits of data. Extra bits at the end of the string are set to zero. Such bits are discarded when creating PDFC from Mars XML content.

Applications creating or manipulating Mars can work with the binary values but will need to convert the base64 representation to binary when extracting the data from or placing it in the Mars XML document.

D.3.4 pdf:doc_encoded_string

This type is used for a string whose representation is known to be in **PDFDocEncoding**. Characters can be converted to and from Unicode using the mapping for PDFDocEncoding.

D.3.5 pdf:text_name

PDFC to Mars

Within a PDFC document, names with this schema type always are UTF-8 encoded. There may or may not be a byte-order marker. Characters should then be transcoded to the appropriate encoding for the generated XML.

The special encoding attribute is not generated for names of this schema type.

Mars To PDFC

When reading XML and generating PDFC, the XML characters should be transcoded to **UTF-8** if necessary.

Processing Text Names in XML

There are no special issues to consider when processing strings of type pdf:text_name in the XML.

D.3.6 pdf:name

Bytes in names of this type may be arbitrary and may not represent characters at all. Usually, such names are single byte character data, but this may not be the case. The objective in processing these names is to make them as useful in XML as possible.

PDFC to Mars

The encoding of the pdf:name object is determined based on the following algorithm.

Algorithm 3 ***Identifying the pdf:name encoding***

1. If the name contains only bytes with values Tab 0x09, LF 0x0A, CR 0x0D, and 0x20-0x7F, then the encoding is **ASCII**.
2. if the name contains only legal byte sequences for UTF-8, then the encoding is **UTF-8**.
3. if an implementation has additional ways to recognize the encoding, they can be applied at this point. If the encoding is positively recognized, that encoding is the result.
4. if none of the above apply, and the name contains only bytes defined by the **PDFDocEncoding**, then the encoding is **PDFDocEncoding**.
5. if none of the above apply, the name is considered binary and the encoding is **BASE64**. Special case: if the bytes are to be used as an attribute value and the string contains leading spaces, trailing spaces, or consecutive internal spaces, then the **BASE64** encoding must be used.

If the encoding determined by the above is other than **ASCII** or **UTF-8**, then a special attribute naming the encoding must be emitted. If the value is going into an XML element, the attribute name is "_enc". If the value is going into an XML attribute, the attribute name containing the encoding value is the attribute name being generated for the string with "_enc" appended.

The value of the _enc attribute must be "**BASE64**" or conform to the character set encoding name guidelines given as part of the XML 1.0 specification section 4.3.3 (namely that it be one of "UTF-8", "UTF-16", "ISO-10646-UCS-2", "ISO-10646-UCS-4", "ISO-8859-1", "ISO-8859-2", ... "ISO-8859-n" (where n is the part number), "ISO-2022-JP", "Shift_JIS", "EUC-JP" or that it be a registered name from the Internet Assigned Numbers Authority [See http://www.iana.org/assignments/character-sets.]). Note that this is not the encoding of the value of the Name attribute (which is specified as part of the encoding declaration in the XML). Rather it is the encoding that was used or that should be used when the file name is converted from or to the byte string stored within the PDF file itself.

Values other than **ASCII**, **PDFDocEncoding**, **UTF-8**, and **UTF-16** should not be used with the expectation that they will be recognized and processed by all tools and implementations.

Example: If the encoding for a pdf:name is identified as UTF-16 and the name itself was to be the value of an element with tag <Value>, then the effect of these rules is to emit an element with tag <Value _enc="UTF-16">.

Another example: If an attribute Filename is being created from a pdf:name whose encoding is determined to be PDFDocEncoding, then a second attribute Filename_enc="PDFDocEncoding" should be generated.

Mars to PDFC

The XML string is mapped to the PDFC name as outlined in "Overview of the string conversion process" on page 154. For the transcoding step, the containing XML element is examined for presence of a "_enc" attribute or a "name_enc" attribute if the string value itself is coming from attribute "name". If present, then this is the target encoding for the string when it is stored in the PDF name. If absent, **UTF-8** (which covers **ASCII** as a subset) is the target encoding for the Unicode string.

Processing Names in XML

Creators of content can control the transcoding of Unicode character values by adding the "_enc" attribute.

D.3.7 pdf:tag_name Used As An XML Tag

PDFC to Mars

The encoding of the pdf:tag_name object is determined as follows:

1. The bytes in the name are inspected to check that they form a legal UTF-8 string. If so, the encoding is UTF-8.
2. If not, the bytes of the name are considered to be 8-bit Unicode character values each in the range 0 to 255. If a byte begins a UTF-8 multi-byte sequence but the following bytes are not legal for that sequence, or the byte is illegal as a lone byte in UTF-8, then the byte is mapped to the range U+E000 to U+E0FF by adding 0xE000 to the byte value.

To convert the resulting name into a legal XML tag name, all tag illegal characters are converted to a partially escaped-hex Unicode representation as defined in section “9.1 Mapping SQL <identifier>s to XML Names” of ISO/IEC JTC 1/SC 32. This effectively maps illegal tag characters to _xDDDD_ where DDDD is the hex unicode value for the character being replaced.

Mars to PDFC

When the XML tag name is to be mapped back to a PDFC name, the XML string is processed character-by-character and inspected for escape sequences. Those sequences are turned back into their Unicode equivalent characters. If the resulting character is in the range U+E000 to U+E0FF then the character is mapped to a byte value in range 00-FF by subtracting 0xE000. The PDFC name is then formed from the **UTF-8** representation of the Unicode string. Note that use of the extended characters U+E000 to U+E0FF is specifically designed to recreate illegal UTF-8 byte sequences.

D.3.8 Summary

TABLE 41 Summary of PDFC to Mars XML Text Conversion

SCHEMA TYPE	ENCODING	DETERMINATION OF ENCODING	UNACCEPTABLE CHARACTERS	*_ENC ATTRIBUTE
pdf:text_string	UTF-16	lead bytes FE FF + no illegal multi-byte sequences.	none	no
	PDFDocEncoding	not UTF-16	00 - 08, 0B-0C 0E-17, 7F, 9F, AD	no
pdf:byte_string	ASCII	all bytes in 00-7F	80-FF	no
	UTF-8	lead bytes EF BB BF, no illegal sequences	none	yes
	UTF-16	lead bytes FE FF, no illegal sequences	none	yes
	PDFDocEncoding	no bytes in 00-08, 0B-0C, 0E-17, 7F, 9F, or AD	00-08, 0B-0C, 0E-17, AD	no
	BASE64	none of above	none	yes
pdf:base64_string	BASE64	always used for this type	none	no
pdf:doc_encoded_string	PDFDocEncoding	always used for this type	00 - 08, 0B-0C 0E-17, 7F, 9F, AD	no
pdf:text_name	UTF-8	always used for this type	none	no
pdf:name	ASCII	all bytes in 00-7F	80-FF	no
	UTF-8	no illegal sequences	none	no
	PDFDocEncoding	no bytes in 00-08, 0B-0C, 0E-17, 7F, 9F, or AD	00-08, 0B-0C, 0E-17, 7F, 9F, or AD	yes
	BASE64	none of above, or whitespace issues	none	yes

SCHEMA TYPE	ENCODING	DETERMINATION OF ENCODING	UNACCEPTABLE CHARACTERS	*_ENC ATTRIBUTE
pdf:tag_name	**UTF-8**	no illegal UTF-8 sequences	none; illegal XML name characters escaped	no
	Other	illegal UTF-8 sequences	none; illegal XML name characters escaped; illegal UTF-8 byte sequences mapped	No

Significant leading or trailing space characters in attributes requires BASE64 encoding.

Special XML characters need to be represented as character entities in certain contexts. See Table 40 on page 156.

TABLE 42 Target Encoding When Converting from Mars to PDFC

_ENC ATTR (BELOW) / SCHEMA TYPE (RIGHT)	BYTE_STRING	TEXT STRING	BASE64 STRING	PDFDOCENCODING	TEXT_NAME	NAME
None	PDFDocEncoding	PDFDocEncoding or UTF-16 + BOM	decode into bytes	PDFDocEncoding	UTF-8	UTF-8
BASE64	decode into bytes	illegal	decode into bytes	illegal	illegal	decode into bytes
UTF-16	UTF-16 + BOM	illegal	decode into bytes	illegal	illegal	UTF-16 + BOM
PDFDocEncoding	PDFDocEncoding	illegal	decode into bytes	illegal	illegal	PDFDocEncoding
UTF-8	UTF-8	illegal	decode into bytes	illegal	illegal	UTF-8

D.3.9 White Space Rules for Mars XML

Whitespace is significant for all attribute and Mixed Content elements (defined in the XML spec as elements that can contain character data optionally inter-

spersed with other child elements) except those designated to be of type **token** in the schema. For items of type **token**, full whitespace normalization is performed as described in the RelaxNG and XML Schema specifications (leading and trailing whitespace is removed, and interior contiguous whitespace is replaced with a single space character).

Elements that have Element Content (defined in the XML spec as elements that have no character data as children, only elements) are allowed to have elements optionally separated by whitespace. The whitespace is ignored and need not be preserved by tools.

D.4 Examples

TABLE 43

SCHEMA	PDF	MARS XML
element Desc { pdf:text_string }	(abcd)	<Desc>abcd</Desc>
element Desc { pdf:text_string }	(\376\377\000a\000b\000c)	<Desc>abc</Desc>
element Desc { pdf:text_string }	(\376\377\000a\060\120\000c) FE FF 00 61 30 50 00 63	<Desc>a(some Japanese character)c </Desc> U+(61 3050 63)
element Desc { pdf:byte_string }	(abcdÁ Í) 61 62 63 64 C1 80 CD	<Desc>abcdÁ Í</Desc> U+(61 62 63 64 C1 2022 CD)
element Desc { attribute Info {pdf:byte_string} }	(\357\273\277abcd\342\200\242) EF BB BF 61 62 63 64 E2 80 A2	<Desc Info_enc=”UTF-8” Info=”abcd ”/> U+(61 62 63 64 2022)
element Desc { attribute Info {pdf:byte_string} }	(special) 20 20 73 70 65 63 69 61 6C	<Desc Info_enc=”BASE64” Info = “ICBzcGVjaWFs” />

Appendix E Universal Container Format

E.1 Overview

E.1.1 Purpose and Scope

This specification defines the Universal Container Format. UCF is a general-purpose container technology. It is based on the packaging principles of OCF, the OEBPS Container Format, created by the International Digital Publishing Forum. The OCF specification describes a general-purpose container technology in the context of encapsulating OEBPS publications. While the OCF specification anticipates that the general-purpose container technology it describes will ultimately be used in other bundling applications, the specification itself does not formally separate the generic technology from its use in the context of OEBPS. The goal of this specification is to do just that, specifying a generic container format that can be used by many applications, where OCF itself is one application.

As a general container format, UCF collects a related set of files into a single-file container. UCF can be used to collect files in various document and data formats and for classes of applications. The single-file container enables easy transport of, management of, and random access to, the collection.

UCF defines rules for how to represent an abstract collection of files (the “abstract container”) into physical representation within a Zip archive (the “physical container”). The rules for Zip containers build upon and are backward compatible with the Zip technology used by Open Document Format (ODF) 1.0.

UCF is the RECOMMENDED single-file container technology for all Zip-based Adobe formats. It is designed to provide a set of lightweight constraints on the use of Zip. It includes a set of optional features including digital signatures and encryption. If an UCF-based file format includes this optional functionality, it should follow the UCF specifications for use of these features. For example, not all UCF-based formats will make use of digital signatures. However, if a format does include support for signatures, it should follow the UCF rules for signatures.

This specification borrows heavily from the OCF specification. Where possible, the same language is used, with the permission of the IDPF.

E.1.2 Definitions

ASCII

American Standard Code for Information Interchange – a 7-bit character encoding based on the English alphabet (ANSI X3.4-1986). When used in this document, ASCII refers to the printable graphic characters in the range 33 (decimal) through 126 (decimal) and the nonprintable space character 32 (decimal).

IRI

Internationalized Resource Identifier (http://www.ietf.org/rfc/rfc3987.txt).

UCF

The Universal Container Format defined by this specification.

UCF Container

A container file that is compliant with the format defined in this specification.

UCF User Agent

A combination of hardware and/or software that accepts documents or data packaged in an UCF Container and makes them available to consumer of the content.

ODF

Open Document Format (http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf).

OEBPS

Open eBook Publication Structure (http://www.idpf.org/oebps/oebps1.2/index.htm).

MIME

Multipurpose Internet Mail Extensions (http://www.ietf.org/rfc/rfc2045.txt). “MIME media types” provide a standard methodology for specifying the content type of objects

Mars

An XML representation of PDF.

RFC

Literally “Request For Comments”, but more generally a document published by the Internet Engineering Task Force (IETF). See http://www.ietf.org/rfc.html.

Relax NG

A schema language for XML (http://www.relaxng.org/).

Rootfile

The top-level file of a rendition of a publication; either the “root” from which all other components can be found or the lone file encapsulating the rendition. The OEBPS rootfile is the OEBPS Package file. A PDF file containing the PDF rendition could also be a rootfile.

XML

Extensible Markup Language (http://www.w3.org/TR/xml/).

Zip

A de facto industry standard bundling and compression format (http://www.pkware.com/business_and_developers/developer/appnote).

E.1.3 Relationship to Other Specifications

This specification combines subsets and applications of other specifications. Together, these facilitate the construction, organization, presentation, and unambiguous interchange of electronic documents:

1. The OEBPS Container Format specification (http://www.idpf.org/ocf/ocf1.0).
2. Zip format (http://www.pkware.com/business_and_developers/developer/appnote)

3. The Extensible Markup Language (XML) 1.0 (Fourth Edition) specification (http://www.w3.org/TR/xml/)
4. The Namespaces in XML 1.0 (Second Edition) specification (http://www.w3.org/TR/xml-names/)
5. XML-Signature Syntax and Processing (http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/)
6. XML Encryption Syntax and Processing (http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/)
7. Extensible Metadata Platform (XMP) (http://www.adobe.com/products/xmp/)
8. The Unicode Consortium. The Unicode Standard, Version 5.0.0, defined by: *The Unicode Standard, Version 5.0* (Boston, MA, Addison-Wesley, 2007, ISBN 0-321-48091-0), as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions for the latest version and additional information on versions of the standard and of the Unicode Character Database).
9. Particular MIME media types (http://www.ietf.org/rfc/rfc4288.txt and http://www.iana.org/assignments/media-types/index.html)
10. Open Document Format for Office Applications (Open Document) v1.0 (http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf)

E.1.4 Conformance

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "RECOMMENDED", "MAY", and "OPTIONAL" in this document MUST be interpreted as described in (http://www.ietf.org/rfc/rfc2119.txt).

This section defines conformance requirements for UCF.

E.1.4.1 Conforming Containers

The term “Conforming UCF Abstract Container” indicates an UCF Abstract Container (See Section E.2.2) that conforms to all the relevant conformance criteria defined in this specification. The term “Conforming UCF Zip Container” indicates a Zip archive that conforms to the relevant Zip container conformance

criteria (See Section E.4) and which implements an instance of a Conforming UCF Abstract Container.

In addition to other conformance criteria defined in this specification, a Conforming UCF Abstract Container MUST meet the following conditions:

- All XML files defined by this specification MUST be well-formed (as defined in XML 1.0).
- All XML defined by this specification files MUST be compatible with the XML 1.0 specification (http://www.w3.org/TR/2006/REC-xml-20060816) and the Namespaces in XML specification (http://www.w3.org/TR/REC-xml-names).

These conditions do not apply to files in the container that are not defined by this specification (files other than container.xml, manifest.xml, metadata.xml, signatures.xml, encryption.xml, and rights.xml).

E.1.4.2 Conforming User Agents

The term “Conforming UCF User Agent” indicates an UCF User Agent that supports all of the mandatory features defined by this specification.

An UCF User Agent that does not support all of the features defined in this specification MUST NOT claim to be a Conforming UCF User Agent and SHOULD provide readily available documentation of the subset of features it supports.

E.1.5 Future Directions

It is the intent of the contributors to this specification that subsequent versions of this specification continue in the directions established by the 1.0 release. Specifically:

- Future versions of this specification are expected to improve alignment with OASIS/ODF and IDPF/OCF.
- Any required functionality not present in relevant official standards shall be defined in a manner consistent with its eventual submission to an appropriate standards body as extensions to existing standards.

E.2 UCF Overview

E.2.1 UCF: A General Container Technology

UCF is designed as a general container technology. In particular, UCF is designed to be upwardly compatible with the container technology used in ODF 1.0 such that a future version of ODF might use UCF.

E.2.2 "Abstract Container" vs. "Physical Container"

An "Abstract Container" defines a file system model for the contents of the container. The file system model MUST have a single common root directory for all of the contents of the container. The special files REQUIRED by UCF MUST be included within the META-INF directory that is a direct child of the root directory.

A "Physical Container" holds the physical manifestation of an abstract container. This specification defines how an abstract container MUST be mapped to the following two physical container technologies:

- *File System Container* – The mapping of an Abstract Container to a file system within computer storage media on a specific platform (e.g., a hard disk on a computer or a data CD) MUST be a one-to-one mapping where each directory and file within the abstract container is represented as a directory or file within the file system. Section E.3.3 defines a set of restrictions on file system names intended to allow files to be easily stored in most modern file systems.
- *Zip Container* - The mapping of an Abstract Container to a Zip archive is defined in Section E.4.

If a user agent processed both types of physical container, the contents of an OCF container MUST be processed the same no matter whether using a File System Container or a Zip Container. In both cases, the UCF User Agent ultimately opens the rootfile, from which it can determine how to process the container.

E.2.3 Examples

(This section is informative.)

This section includes an example of a OEBPS and Mars files.

E.2.3.1 Example of an OEBPS file

To illustrate the concepts from the previous section, let's assume we have a single OEBPS 1.0 document of Dickens' "Great Expectations" which consists of an OEBPS 1.2 package file ("Great Expectations.opf") and a large number of HTML files, one for the cover page (e.g., "cover.html") and one for each chapter (e.g., "chapter01.html"). The contents of the *publication* might be as follows:

OEBPS 1.2 Publication:
Great Expectations.opf
cover.html
chapters/
  chapter01.html
  chapter02.html
  … *other HTML files for the remaining chapters* …

The contents of the Abstract Container include all of the assets from the Publication, plus a small number of files defined by UCF within the META-INF directory. Note that container.xml is REQUIRED by OCF even though it is not required by UCF. See Section E.3.5 for descriptions of the files within the META-INF directory.

Abstract Container:
mimetype
META-INF/
  container.xml
  [manifest.xml]
  [metadata.xml]
  [signatures.xml]
  [encryption.xml]
  [rights.xml]
OEBPS/
  Great Expectations.opf
  cover.html
  chapters/
    chapter01.html
    chapter02.html
    … other HTML files for the remaining chapters …

When the above abstract container is mapped to a File System Container, the directory structure within the file system exactly matches the UCF's Abstract Container directory structure shown above:

File System Container:
…some directory within the file system…/
mimetype
META-INF/
  container.xml
  [manifest.xml]
  [metadata.xml]
  [signatures.xml]
  [encryption.xml]
  [rights.xml]
OEBPS/
  Great Expectations.opf
  cover.html
  chapters/
    chapter01.html
    chapter02.html
    … other HTML files for the remaining chapters …

When the above Abstract Container is stored within a Zip container, the contents of the Zip archive will match the directory structure shown above. When the Zip archive is created, the mimetype file must be the first file in the archive. [See Section E.3.4.]

Zip Container:
mimetype
META-INF/
  container.xml
  [manifest.xml]
  [metadata.xml]
  [signatures.xml]
  [encryption.xml]
  [rights.xml]
OEBPS/
  Great Expectations.opf
  cover.html
  chapters/
    chapter01.html
    chapter02.html
    … other HTML files for the remaining chapters …

The mimetype file contains the string “application/epub+zip.”

The corresponding META-INF/container.xml file might appear as follows:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/Great Expectations.opf"
     media-type="application/oebps-package+xml" />
  </rootfiles>
</container>
```

Note: *N.B. The use of the specific namespace string “urn:oasis:names:tc:open-document:xmlns:container” should be considered provisional until approved by an OASIS technical committee.*

E.2.3.2 Example of a Mars file

A Mars file, unlike an OEBPS file, does not require container.xml to include a rootfiles element. The root file is always backbone.xml. The example file does not include digital signatures, encrypted data, or DRM restrictions and therefore does not include signatures.xml, encryption.xml, or rights.xml.

Abstract Container:

```
mimetype
META-INF/
  container.xml
  [manifest.xml]
  [metadata.xml]
backbone.xml
page/0/
  info.xml
  pg.svg
  im_1.png
  f_1.fd
fonts/
  f_1._sfnt
```

The corresponding META-INF/container.xml file might appear as follows:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
 <relationships xmlns:pdf="http://ns.adobe.com/pdf/2006">
  <relationship type="metadata" target="$path.xmp"/>
  <relationship type="pdf:annotation" target="$path.ann"/>
 </relationships> </container>
```

A Mars container.xml file includes relationship elements for metadata and annotations.

E.3 UCF Container Contents

E.3.1 File and directory structure

The virtual file system for the UCF “Abstract Container” MUST have a single common root directory for all of the contents of the container.

The following file names in the root directory are reserved:

- “mimetype”
- “META-INF”

The “mimetype” file is discussed in Section E.4. The META-INF/ directory contains the reserved files used by UCF. These reserved files are described in the following sections. All other files within the Abstract Container MAY be in any location descendant from the root directory except for “mimetype” at the root level or directly within the META-INF directory. An UCF based-format may include files in the META-INF directory as long as these files are within subdirectories in META-INF. The names of these subdirectories MUST NOT include the “.” character. This avoids conflicts with files specified by future versions of the UCF specification. Any file in the META-INF directory used by UCF will include a “.” character.

It is RECOMMENDED that the contents of individual documents or applications be stored within dedicated sub-directories to minimize potential file name collisions in the event that multiple renditions are used or that multiple publications per container are supported in future versions of this Specification.

E.3.2 Relative IRIs for referencing other components

Files within the Abstract Container reference each other via Relative IRI References (http://www.ietf.org/rfc/rfc3987.txt and http://www.ietf.org/rfc/rfc3986.txt), no matter what is used for the physical container (e.g., File System Container or Zip Container). For example, if a file named “chapter1.html” references an image file named “image1.jpg” that is located in the same directory, then “chapter1.html” might contain the following as part of its content:

```
<img src="image1.jpg" alt="…" …/>
```

For Relative IRI References, the Base IRI (see RFC3986) is determined by the relevant language specifications for the given file formats. For example, the CSS specification defines how relative IRI references work in the context of CSS style sheets and property declarations.

Unlike many language specifications, the Base IRIs for all files within the META-INF/ directory use the root folder for the Abstract Container as the default Base IRI. For example, if META-INF/container.xml has the following content:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/Great Expectations.opf"
     media-type="application/oebps-package+xml" />
  </rootfiles>
</container>
```

the path “OEBPS/Great Expectations.opf” is relative to the root directory for the Abstract Container and *not* relative to the META-INF/ directory.

If a Relative IRI Reference contains an absolute path (an IRI that has no schema or authority but begins with a “/”), the reference is resolved relative to the root directory of the Abstract Container. For example, in the example in Section E.2.3.1, the IRI “/OEBPS/cover.html” will refer to the file OEBPS/cover.html no matter what file the IRI is found in.

E.3.3 File Names

The term File Name represents the name of any type of file, either a directory or an ordinary file within a directory within an Abstract Container. For a given directory within the Abstract Container, the Path Name is a string holding all directory names in the full path concatenated together with a “/” character separating the directory names. For a given file within the Abstract Container, the Path Name is the string holding all directory names concatenated together with a “/” character separating the directory names, followed by a “/” character and then the name of the file. The File Name restrictions described below are designed to allow directory names and file names to be used without modification on most commonly used operating systems. This specification does not specify how a UCF User Agent that is unable to represent UCF conforming File Names would compensate for this incompatibility.

The following statements apply to Conforming UCF Content:

- File Names MUST be UTF-8 encoded with the restrictions below
- When represented as UTF-8, File Names MUST NOT exceed 255 bytes
- When represented as UTF-8, the Path Name for any directory or file within the Abstract Container MUST NOT exceed 65535 bytes
- File Names MUST NOT use the following characters (These characters are not be supported always across commonly used operating systems):
    - U+0022 " QUOTATION MARK
    - U+002A * ASTERISK
    - U+002E . FULL STOP, as the last character
    - U+002F / SOLIDUS
    - U+003A : COLON
    - U+003C < LESS-THAN SIGN
    - U+003E > GREATER-THAN SIGN
    - U+003F ? QUESTION MARK
    - U+005C \ REVERSE SOLIDUS
    - The C0 controls, U+0000 through U+001F and U+007F
- File Names are case sensitive.

- Two File Names within the same directory MUST NOT map to the same string following case normalization (http://www.unicode.org/reports/tr21/tr21-5.html). Two File Names that differ only in case are disallowed within the same directory.
- Two File Names within the same directory MUST NOT be canonically equivalent in the Unicode sense.

Note that some commercial Zip tools do not support the full Unicode range and may only support the ASCII range for File Names. Content creators who want to use Zip tools that have these restrictions MAY find it is best to restrict their File Names to the ASCII range. If the names of files can not be preserved during the unzipping process, it will be necessary to compensate for any name translation which took place when the files are referenced by URI from within the content.

E.3.4 Container media type identification

It is frequently necessary for applications to determine the media type of a file. This is usually accomplished by looking at the file extension of the file. This gives applications a quick way to determine the type of the file without looking inside the file. UCF Container files SHOULD use an extension specific to the kind of UCF Container it is.

Unfortunately, the identification of files through the use of file extensions is notoriously unreliable. As a result, it is desirable to have a more robust way of identifying files independent of their file names or extensions. One mechanism that has evolved for doing this is to require the placement of specific information at specific file offsets. A processing agent can then check a fixed location to determine if the file is a specific type of UCF Container.

The method that has evolved for doing this in Zip archives is the inclusion of an uncompressed, unencrypted file called “mimetype” as the first file in the Zip archive. The contents of this file are the media type of the file. UCF Containers MUST place the media type as an ASCII string in the “mimetype” file as the first file in the Zip archive. See Section E.4 for more detail on this mechanism.

E.3.5 META-INF

All valid UCF Containers MAY include a directory called “META-INF” at the root level of the container file system. This directory contains the files specified

below that describe the contents, metadata, signatures, encryption, rights and other information about the contained publication.

The semantics of the following files that MAY be present at the “META-INF/” level are specified. All other files found at the “META-INF/” level MUST be ignored by conformant UCF User Agents.

E.3.5.1 Container – META-INF/container.xml (Optional)

(This is normative.)

An UCF Container MAY include a file called “container.xml” within the “META-INF” directory at the root level of the container file system. If present, the container.xml file MAY identify the MIME type of, and path to, the rootfile for the container and any OPTIONAL alternate renditions included within the container. An UCF-based format MUST either require container.xml to identify the rootfile or specify a format-specific method for initiating processing of the container. The container.xml file MAY specify implicit relationships in the container, as described in Section 3.5.1.2.

The container.xml file MUST NOT be encrypted.

The container.xml file contains XML that uses the “urn:oasis:names:tc:opendocument:xmlns:container” namespace for all of its elements and attributes. The “version="1.0"” attribute MUST be included for all containers that conform to this version of the specification.

A RELAX NG UCF schema describing the <container> element that MUST be the root element of container.xml can be found in the Section E.5.

E.3.5.1.1 Rootfiles (Optional)

The <rootfiles> element MUST contain at least one <rootfile> element.

Each <rootfile> element specifies the rootfile of a single rendition of the contained publication. A rootfile often includes an enumeration of the other files needed by the rendition.

(This example is informative.)

The following example shows a sample container.xml for an UCF container inside of which is an OEBPS Publication with the root file “OEBPS/My%20Crazy%20Life.opf” (the OEBPS package file):

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/My%20Crazy%20Life.opf"
     media-type="application/oebps-package+xml" />
  </rootfiles>
</container>
```

(This example is informative.)

The following example adds an alternate PDF version of the Publication:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/My%20Crazy%20Life.opf"
     media-type="application/oebps-package+xml" />
    <rootfile full-path="PDF/My%20Crazy%20Life.pdf"
     media-type="application/pdf" />
  </rootfiles>
</container>
```

(This is normative.)

The values of the full-path attributes MUST contain a “path component” (as defined by RFC3986) which MUST only take the form of a “path-rootless” (as defined by RFC3986). The path components are relative to the root of the container in which they are used.

Conforming UCF User Agents MUST ignore unrecognized elements (and their contents) and unrecognized attributes within a container.xml file, including unrecognized elements and unrecognized attributes from other namespaces.

Conforming container.xml files MUST be valid according to the RELAX NG UCF schema with the <container> element as the root element after removing all elements (and child nodes of these elements) and attributes from other namespaces.

(This example is informative.)

For example:

```
<?xml version="1.0"?>
<container version="1.0"
 xmlns="urn:oasis:names:tc:opendocument:xmlns:container"
 foo:xmlns="..."
 foozle:xmlns="..." />
  <foo:bar />
  <rootfiles foozle:identifier="bar">
  ...
  </rootfiles>
</container>
```

is conformant, but:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <foo />
  <rootfiles>
  ...
  </rootfiles>
</container>
```

is non-conformant due to the non-namespace-qualified use of the <foo> element.

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles identifier="bar">
  ...
  </rootfiles>
</container>
```

is also non-conformant due to the non-namespace-qualified use of the “identifier” attribute on the <rootfiles> element.

E.3.5.1.2 Relationships (Optional)

Container.xml MAY include information that identifies implicit relationships between files in a container. Usually, one file explicitly references another file. For example, an SVG page description may reference an image. At times, it is conve-

nient to indirectly reference a file. For example, one file may have metadata associated with it stored in a second file. Using an indirect association makes it possible to add metadata by simply creating a metadata resource and not changing the original resource or, in fact, any resource in the package.

Beyond convenience, indirect associations enable container files to be digitally signed while still permitting the addition of certain kinds of data such as metadata or annotations. For example, a document with no annotations can be signed. Annotations can then be added without invalidating the signature.

UCF provides a generic mechanism for establishing a relationship between two container files. A relationship specifies a relationship type and a mapping from a set of source names to target names. For any given file, this relationship can be used to determine the related file. However, UCF does not require that the related file exist.

The root <container> element MAY contain a child <relationships> element. This element MAY contain one or more child <relationship> elements that describe specific relationships. Each relationship element includes attributes type and target that specify a relationship type and a pattern that maps source to target names. The type is a qualified name. UCF defines one relationship type, "metadata." UCF-based formats can add other types by specifying a namespace. The target pattern is a string that may include the following variables that are substituted when a relationship is resolved:

Variable	Definition	Example
path	the full path to the resource	/a/b/c.d
dir	the directory (without the filename)	/a/b
filename	the path without the directory	c.d
basename	the filename without the extension	c
ext	the filename extension	d

These variables are specified by enclosing their name in braces and preceding the opening brace with a "$". Braces may be omitted if the first character after the variable name is not a letter. A target is resolved by copying the ordinary text in the pattern and replacing the variables with their values.

For example:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <relationships xmlns:pdf="http://ns.adobe.com/pdf/2006">
    <relationship type="metadata" target="$path.xmp"/>
    <relationship type="pdf:annotation" target="$path.ann"/>
  </relationships>
</container>
```

The first relationship specifies that any file may have associated metadata. This metadata is found in a file that has the same name as the original with ".xmp" added as an extension. For example, /page/p0/pg.svg may have metadata in /page/p0/pg.svg.xmp. The second relationship specifies that any file may have associated annotations. The annotations are found in a file that has ".ann" added as an extension.

E.3.5.2 Manifest – META-INF/manifest.xml (Optional)

An OPTIONAL file with the reserved name "manifest.xml" within the "META-INF" directory at the root level of the container may appear in a valid UCF container. If present, the file's content MUST be as defined in the ODF 1.0 manifest schema (http://www.oasis-open.org/committees/download.php/12570/OpenDocument-manifest-schema-v1.0-os.rng).

The manifest.xml file, if present, MUST NOT be encrypted.

E.3.5.3 Metadata – META-INF/metadata.xml (Optional)

A file with the reserved name "metadata.xml" within the "META-INF" directory at the root level of the container file system may appear in a valid UCF container. This file, if present, MUST be used for container-level metadata. In version 1.0 of OCF, no such container-level metadata is specified.

If the "META-INF/metadata.xml" file exists, its contents MUST be valid XML with namespace-qualified elements to avoid collision with future versions of OCF that MAY specify a particular grammar and namespace for elements and attributes within this file.

Adobe-defined formats based on UCF MUST use XMP to specify metadata (http://www.adobe.com/products/xmp/).

E.3.5.4 Digital Signatures – META-INF/signatures.xml (Optional)

An OPTIONAL "signatures.xml" file within the "META-INF" directory at the root level of the container file system holds digital signatures of the container and its contents. The contents of this file is not specified in UCF 1.0. However, a future revision of UCF will define the format for this file. See Section E.8 for the likely definition.

E.3.5.5 Encryption – META-INF/encryption.xml (Optional)

An OPTIONAL "encryption.xml" file within the "META-INF" directory at the root level of the container file system holds all encryption information on the contents of the container. The contents of this file is not specified in UCF 1.0. However, a future revision of UCF will define the format for this file. See Section E.9 for the likely definition.

E.3.5.6 Rights Management – META-INF/rights.xml (Optional)

An OPTIONAL file with the name "rights.xml" within the "META-INF" directory at the root level of the container file system is a reserved name in a valid UCF container. This location is reserved for digital rights management (DRM) information for trusted exchange of Publications among rights holders, intermediaries, and users. In version 1.0 of UCF, there is not a REQUIRED format for DRM information, but a future version of this specification MAY specify a particular format for DRM information.

If the "META-INF/rights.xml" file exists, it MUST be a well-formed XML document which uses and conforms to XML Namespaces it uses, and its contents SHOULD be valid XML with namespace-qualified elements to avoid collision with future versions of UCF that MAY specify a particular format this file.

The rights.xml file MUST NOT be encrypted.

When the rights.xml file is not present, the UCF container provides no information indicating any part of the container is rights governed.

E.4 Zip Container

UCF's Zip Container supports the Zip format as specified by the application note at http://www.pkware.com/business_and_developers/developer/appnote/, but with the following constraints and clarifications:

- Conforming UCF Zip Containers MUST NOT use the features in the Zip application note that allow Zip files to be split across multiple storage media. Conforming UCF User Agents MUST treat any UCF files that specify that the Zip file is split across multiple storage media as being in error.
- Conforming UCF Zip Containers MUST only include uncompressed files or Flate-compressed files within the Zip archive. Conforming UCF User Agents MUST treat any UCF Containers that use compression techniques other than Flate as being in error.
- Conforming UCF Zip Containers MAY use the Zip64 extensions and SHOULD only use those extensions when the content requires them. Conforming UCF User Agents MUST support the Zip64 extensions.
- Conforming UCF Zip Containers MUST NOT use the encryption features defined by the Zip format; instead, encryption MUST be done using the features described in Section E.9. Conforming UCF User Agents MUST treat any other UCF Zip Containers that use Zip encryption features as being in error.
- It is not a requirement that Conforming UCF User Agents preserve information from an UCF Zip Container through load and save operations that do not map to corresponding representation within the UCF Abstract Container; in particular, a Conforming UCF User Agent does not have to preserve CRC values, comment fields or fields that hold file system information corresponding to a particular operating system (e.g., "External file attributes" and "Extra field")
- Conforming UCF Zip Containers MUST encode File System Names using UTF-8.

Here are some details about particular fields in the Zip archive:

- On the local file header table, Conforming UCF Zip Containers MUST set the 'version needed to extract' fields to the values 10, 20 or 45 in order to match the maximum version level needed by the given file (e.g., 20 if Deflate is need-

ed, 45 if Zip64 is needed). Conforming UCF User Agents MUST treat any other values as being in error.

- On the local file header table, Conforming UCF Zip Containers MUST set the 'compression' method field to the values 0 or 8. Conforming UCF User Agents MUST treat any other values as being in error.
- Conforming UCF User Agents MUST treat UCF Zip Containers with an "Archive decryption header" or an "Archive extra data record" as being in error.

The first file in the Zip Container MUST be a file by the ASCII name of 'mimetype' which holds the MIME type for the Zip Container (i.e., "application/epub+zip" as an ASCII string; no padding, white-space or case change). The file MUST be neither compressed nor encrypted and there MUST NOT be an extra field in its Zip header. If this is done, then the Zip Container offers convenient "magic number" support as described in RFC 2048 and the following will hold true:

- The bytes "PK" will be at the beginning of the file
- The bytes "mimetype" will be at position 30
- The actual MIME type (i.e., the ASCII string "application/epub+zip") will begin at position 38

E.5 RELAX NG UCF Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<choice xmlns="http://relaxng.org/ns/structure/1.0">
  <element name="container">
    <attribute name="version">
      <value>1.0</value>
    </attribute>
    <attribute name="xmlns">
      <value>urn:oasis:names:tc:opendocument:xmlns:container</value>
    </attribute>
  <optional>
    <element name="rootfiles">
      <oneOrMore>
        <element name="rootfile">
          <attribute name="full-path">
```

```
          <text/>
         </attribute>
         <attribute name="media-type">
          <text/>
         </attribute>
        </element>
       </oneOrMore>
      </element>
     </optional>
    <optional>
     <element name="relationships">
      <oneOrMore>
       <element name="relationship">
        <attribute name="type">
         <text/>
        </attribute>
        <attribute name="target">
         <text/>
        </attribute>
       </element>
      </oneOrMore>
     </element>
    </optional>
   </element>

   <element name="signatures">
    <attribute name="xmlns">
     <value>urn:oasis:names:tc:opendocument:xmlns:container</value>
    </attribute>
    <oneOrMore>
     <element name="Signature" ns="http://www.w3.org/2001/04/xmldsig#">
      <externalRef
       href="http://www.w3.org/Signature/2002/07/xmldsig-core-schema.rng"/>
     </element>
    </oneOrMore>
   </element>

   <element name="encryption">
    <attribute name="xmlns">
     <value>urn:oasis:names:tc:opendocument:xmlns:container</value>
    </attribute>
    <oneOrMore>
     <choice>
```

```
      <element name="EncryptedData" ns="http://www.w3.org/2001/04/xmlenc#">
        <externalRef
          href="http://www.w3.org/Encryption/2002/07/xenc-schema.rng"/>
      </element>
      <element name="EncryptedKey" ns="http://www.w3.org/2001/04/xmlenc#">
        <externalRef
          href="http://www.w3.org/Encryption/2002/07/xenc-schema.rng"/>
      </element>
    </choice>
  </oneOrMore>
 </element>

</choice>
```

E.6 Example

The following example demonstrates the use of this UCF format to contain a signed and encrypted OEBPS publication with an alternate PDF rendition within a Zip Container.

Ordered list of files in the Zip Container:

mimetype
META-INF/container.xml
META-INF/signatures.xml
META-INF/encryption.xml
OEBPS/As You Like It.opf
OEBPS/book.html
OEBPS/images/cover.png
PDF/As You Like It.pdf

The `mimetype` file:

application/epub+zip

The `META-INF/container.xml` file:

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
 <rootfiles>
  <rootfile full-path="OEBPS/As You Like It.opf"
```

```
        media-type="application/oebps-package+xml" />
   <rootfile full-path="OEBPS/As You Like It.pdf"
        media-type="application/pdf" />
  </rootfiles>
</container>
```

The `META-INF/signatures.xml` file:

```
<?xml version="1.0"?>
<signatures
xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <Signature Id="AsYouLikeItSignature" xmlns="http://
www.w3.org/2000/09/xmldsig#">

    <!-- SignedInfo is the information that is actually signed.
In this case -->
    <!-- the SHA1 algorithm is used to sign the canonical form
of the XML    -->
  <!-- documents enumerated in the Object element below
-->
    <SignedInfo>
      <CanonicalizationMethod
       Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>

    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>

    <Reference URI="#AsYouLikeIt">

     <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

     <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
   </Reference>
  </SignedInfo>

  <!-- The signed value of the digest above using the DSA algorithm -->

  <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>

  <!-- The key to use to validate the signature -->
  <KeyInfo>
   <KeyValue>
    <DSAKeyValue>
```

```
        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>

  <!-- The list documents to sign. Note that the canonical form of XML  -->
  <!-- documents is signed while the binary form of the other documents -->
  <!-- is used -->
  <Object>
    <Manifest Id="AsYouLikeIt">
      <Reference URI="OEBPS/As You Like It.opf">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        </Transforms>
      </Reference>
      <Reference URI="OEBPS/book.html">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        </Transforms>
      </Reference>
      <Reference URI="OEBPS/images/cover.png" />
      <Reference URI="PDF/As You Like It.pdf" />
    </Manifest>
  </Object>

 </Signature>
</signatures>
```

The `META-INF/encryption.xml` file:

```
<?xml version="1.0"?>
<encryption
 xmlns="urn:oasis:names:tc:opendocument:xmlns:container"
 xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

  <-- The RSA encrypted AES-128 symmetric key used to encrypt
the data -->
  <enc:EncryptedKey Id="EK”>
    <enc:EncryptionMethod  Algorithm="http://www.w3.org/2001/
04/xmlenc#rsa-1_5"/>
    <ds:KeyInfo>
      <ds:KeyName>John Smith</ds:KeyName>
    </ds:KeyInfo>
```

```
    <enc:CipherData>
      <enc:CipherValue>xyzabc...</enc:CipherValue>
    </enc:CipherData>
  </enc:EncryptedKey>

  <!-- Each EncryptedData block identifies a single document
       that has been encrypted using the AES-128 algorithm. The
       data remains stored in it's encrypted form in the
       original file within the container.       -->
  <enc:EncryptedData Id="ED1">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
  <ds:KeyInfo>
   <ds:RetrievalMethod URI="#EK"
    Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
  </ds:KeyInfo>
  <enc:CipherData>
   <enc:CipherReference URI="OEBPS/book.html"/>
  </enc:CipherData>
 </enc:EncryptedData>

  <enc:EncryptedData Id="ED2">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
  <ds:KeyInfo>
   <ds:RetrievalMethod URI="#EK"
    Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
  </ds:KeyInfo>
  <enc:CipherData>
   <enc:CipherReference URI="OEBPS/images/cover.png"/>
  </enc:CipherData>
 </enc:EncryptedData>

  <enc:EncryptedData Id="ED3">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
  <enc:KeyInfo>
   <enc:RetrievalMethod URI="#EK"
    Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
  </enc:KeyInfo>
  <enc:CipherData>
   <enc:CipherReference URI="PDF/As You Like It.pdf"/>
  </enc:CipherData>
 </enc:EncryptedData>

</encryption>
```

The `OEBPS/As You Like It.opf` file:

```
<?xml version="1.0"?>
<!DOCTYPE package PUBLIC "+//ISBN 0-9673008-1-9//DTD OEB 1.2 Package//EN"
```

```
 "http://openebook.org/dtds/oeb-1.2/oebpkg12.dtd">
<package unique-identifier="Package-ID">
 <metadata>
  <dc-metadata xmlns:dc="http://purl.org/dc/elements/1.0"
            xmlns:oebpackage="http://openebook.org/namespaces/oeb-package/1.0">
   <dc:Identifier id="Package-ID">ebook:guid-6B2DF0030656ED9D8</dc:Identifier>
   <dc:Title>As You Like It</dc:Title>
   <dc:Creator role="aut">William Shakespeare</dc:Creator>
   <dc:Identifier>0-7410-1455-6</dc:Identifier>
   <dc:Subject></dc:Subject>
   <dc:Type></dc:Type>
   <dc:Date event="publication">3/24/2000</dc:Date>
   <dc:Date event="copyright">1/1/9999</dc:Date>
   <dc:Identifier scheme="ISBN">0-7410-1455-6</dc:Identifier>
   <dc:Publisher>Project Gutenberg</dc:Publisher>
   <dc:Language></dc:Language>
  </dc-metadata>
 </metadata>
 <manifest>
  <item id="4915" href="book.html" media-type="text/x-oeb1-document"/>
  <item id="7184" href="images/cover.png" media-type="image/png" />
 </manifest>
 <spine>
  <itemref idref="4915"/>
 </spine>
</package>
```

The `OEBPS/book.html` file:

This file would be binary and be encrypted. Its decrypted contents might look something like:

```
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC
 "+//ISBN 0-9673008-1-9//DTD OEB 1.2 Document//EN"
 "http://openebook.org/dtds/oeb-1.2/oebdoc12.dtd">
<html>
<head>

 ...

</head>
<body>
```

```
...

<img src="images/cover.png" alt="Cover image: a picture of the Bard of Avon" />
 ...
</body>
</html>
```

The `OEBPS/images/cover.png` file:

This file contains the encrypted binary bytes of the cover.png file.

The `OEBPS/As You Like It.pdf` file:

This file contains the encrypted binary bytes of the PDF file.

E.7 Comparison of UCF and OCF

As described in the introduction, UCF is OCF without the OEBPS dependencies. There are only a few significant differences:

- The OCF specification states that the media type of the container must be application/epub+zip, while UCF specifies that UCF-based formats should choose an appropriate media type. OCF implicitly encourages the use of "+zip" to identify Zip-based formats.
- OCF requires all XML documents in a container to be compatible with XML 1.1. UCF requires all XML documents defined by this specification to be compatible with XML 1.0. (The use of XML 1.0 follows generally recommended practice to use XML 1.0 rather than XML 1.1 unless XML 1.1 features are required.)
- OCF forbids certain characters in names. In addition to those characters, UCF also disallows characters corresponding to the non-printing ASCII codes. While the OCF specifications lists forbidden characters by ASCII names, this specification lists Unicode code points.
- OCF requires that two file names in a container be unique after case normalization. UCF also requires that names be unique after character normalization using Unicode normalization form C (canonical decomposition followed by canonical composition).

- OCF requires container.xml which in turn requires specification of a rootfile. UCF does not require container.xml. The rationale is that an UCF-based format can specify how to begin processing the container. For example, Mars always uses backbone.xml. This eliminates the need to find, read, and process an extra file when opening a Mars file, which can have significant cost when viewing Mars files on the Web.
- UCF adds file relationships, providing an application-independent method for storing and finding metadata associated with container files.
- OCF specifies that the metadata.xml file must not be encrypted, while UCF allows this. Even when a publication is protected with encryption (usually to support digital rights management), the IDPF wants reading systems to be able to provide users with useful metadata. Mars (and possibly other UCF formats as well) has more general requirements and needs to leave this as an option for the container creator.
- OCF encourages but does not require each publication (in UCF, generalized to document or application) to reside in its own directory within the container. This makes it easier to contain multiple renditions of the publication in the container.
- The UCF specification has deferred the definition of encryption and digital signature features to a future revision. This will provide implementers more time to evaluate the proposed definition.

E.8 Digital Signatures

Support of digital signatures in UCF has been deferred until a future revision of the specification. However, it is likely that the specification of this feature will match the OCF specification, which follows:

An OPTIONAL “signatures.xml” file within the “META-INF” directory at the root level of the container file system holds digital signatures of the container and its contents. This file is an XML document whose root element is <signatures>. The <signatures> element contains child elements of type <Signature> as defined by “XML-Signature Syntax and Processing” (http://www.w3.org/TR/2002/REC-xmldsig-core-20020212). Signatures can be applied to the publication and any alternate renditions as a whole or to parts of the publication and renditions. XML Signature can specify the signing of any kind of data, not just XML.

The signatures.xml file MUST NOT be encrypted.

When the signatures.xml file is not present, the UCF container provides no information indicating any part of the container is digitally signed at the container level. It is however possible that digital signing exists within any optional alternate contained renditions.

A RELAX NG UCF schema describing the <signature> element that MUST be the root element of signatures.xml can be found in the Section E.5.

When an UCF agent creates a signature of data in a container, it SHOULD add the new signature as the last child <Signature> element of the <signatures> element in the signatures.xml file.

Each <Signature> in the signatures.xml file identifies by IRI the data to which the signature applies, using the XML Signature <Manifest> element and its <Reference> sub-elements. Individual contained files MAY be signed separately or together. Separately signing each file creates a digest value for the resource that can be validated independently. This approach MAY make a Signature element larger. If files are signed together, the set of signed files can be listed in a single XML Signature <Manifest> element and referenced by one or more <Signature> elements.

Any or all files in the container can be signed in their entirety with the exception of the signatures.xml file since that file will contain the computed signature information. Whether and how the signatures.xml file SHOULD be signed depends on the objective of the signer.

- If the signer wants to allow signatures to be added or removed from the container without invalidating the signer's signature, the signatures.xml file SHOULD NOT be signed.
- If the signer wants any addition or removal of a signature to invalidate the signer's signature, the Enveloped Signature transform (defined in Section 6.6.4 of "XML-Signature Syntax and Processing") can be used to sign the entire pre-existing signature file excluding the <Signature> being created. This transform would sign all previous signatures, and it would become invalid if a subsequent signature was added to the package.
- If the signer wants the removal of an existing signature to invalidate the signer's signature but also wants to allow the addition of signatures, an XPath

transform can be used to sign just the existing signatures. (This is only a suggestion. The particular XPath transform is not a part of UCF specification.)

XML-Signature does not associate any semantics with a signature, however an agent MAY include semantic information, for example, by adding information to the Signature element that describes the signature. XML Signature describes how additional information can be added to a signature (for example, by using the SignatureProperties element).

(This example is informative.)

The following XML expression shows the content of an example "signatures.xml" file, and is based on the examples found in Section 2 of "XML-Signature Syntax and Processing." It contains one signature, and the signature applies to two resources, OEBFPS/book.html and OEBFPS/images/cover.jpeg, in the container.

```
<signatures>
 <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
   <CanonicalizationMethod
    Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
   <Reference URI="#Manifest1">
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
   </Reference>
  </SignedInfo>
  <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
  <KeyInfo>
   <KeyValue>
    <DSAKeyValue>
     <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
    </DSAKeyValue>
   </KeyValue>
  </KeyInfo>
  <Object>
   <Manifest Id="Manifest1">
    <Reference URI="OEBFPS/book.xml">
     <Transforms>
      <Transform
       Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
     </Transforms>
    </Reference>
    <Reference URI="OEBFPS/images/cover.jpeg"/>
   </Manifest>
```

```
</Object>
</Signature>
</signatures>
```

E.9 Encryption

Support of encryption in UCF has been deferred until a future revision of the specification. With one exception, it is likely that the specification of this feature will match the OCF specification, which follows this paragraph. The exception is that it will be possible to specify that a particular algorithm does not require Flate compression of data before encryption.

An OPTIONAL “encryption.xml” file within the “META-INF” directory at the root level of the container file system holds all encryption information on the contents of the container. This file is an XML document whose root element is <encryption>. The <encryption> element contains child elements of type <EncryptedKey> and <EncryptedData> as defined by “XML Encryption Syntax and Processing” (http://www.w3.org/TR/2002/REC-xmlenc-core-20021210). Each EncryptedKey element describes how one or more container files are encrypted. Consequently, if any resource within the container is encrypted, “encryption.xml” MUST be present to indicate that the resource is encrypted and provide information on how it is encrypted.

An <EncryptedKey> element describes each encryption key used in the container, while an <EncryptedData> element describes each encrypted file. Each <EncryptedData> element refers to an <EncryptedKey> element, as described in XML Encryption.

A RELAX NG UCF schema describing the <encryption> element that MUST be the root element of encryption.xml can be found in the Section E.5.

When the encryption.xml file is not present, the UCF container provides no information indicating any part of the container is encrypted.

UCF encrypts individual files independently, trading off some security for improved performance, allowing the container contents to be incrementally decrypted. Encryption in this way still exposes the directory structure and file naming of the whole package.

UCF uses XML Encryption to provide a framework for encryption, allowing a variety of algorithms to be used. XML Encryption specifies a process for encrypting arbitrary data and representing the result in XML. Even though an UCF container MAY contain non-XML data, XML Encryption can be used to encrypt all data in an UCF container. UCF encryption supports only encryption of whole files. The encryption.xml file, if present, MUST NOT be encrypted.

Encrypted data replaces unencrypted data in an UCF container. For example, if an image named “photo.jpeg” is encrypted, the contents of the photo.jpeg resource SHOULD be replaced by its encrypted contents. When stored in a Zip container, files MUST be compressed before they are encrypted; Flate compression MUST be used. Within the Zip local file header and central directory, encrypted files SHOULD be listed as stored rather than Flate-compressed.

The following files MUST never be encrypted (regardless of whether default or specific encryption is requested):

- mimetype
- META-INF/container.xml
- META-INF/manifest.xml
- META-INF/metadata.xml
- META-INF/signatures.xml
- META-INF/encryption.xml
- META-INF/rights.xml

Signed resources MAY subsequently be encrypted by using the Decryption Transform for XML Signature. This feature enables an application such as an UCF agent to distinguish data that was encrypted before signing from data that was encrypted after signing. Only data that was encrypted after signing MUST be decrypted before computing the digest used to validate the signature.

(This example is informative.)

In the following example, adapted from Section 2.2.1 of “XML Encryption Syntax and Processing,” the resource image.jpeg is encrypted using a symmetric key algorithm (AES) and the symmetric key is further encrypted using an asymmetric key algorithm (RSA) with a key of John Smith.

<encryption

```
xmlns ="urn:oasis:names:tc:opendocument:xmlns:container"
xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
 <enc:EncryptedKey Id="EK">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <ds:KeyInfo>
   <ds:KeyName>John Smith</ds:KeyName>
  </ds:KeyInfo>
  <enc:CipherData>
   <enc:CipherValue>xyzabc</enc:CipherValue>
  </enc:CipherData>
 </enc:EncryptedKey>
 <enc:EncryptedData Id="ED1">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
  <ds:KeyInfo>
   <ds:RetrievalMethod URI="#EK"
    Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
  </ds:KeyInfo>
  <enc:CipherData>
   <enc:CipherReference URI="image.jpeg"/>
  </enc:CipherData>
 </enc:EncryptedData>
</encryption>
```

Appendix F

This section intentionally left blank.