Adobe® Supplement to the ISO 32000

BaseVersion: 1.7
ExtensionLevel: 3

Adobe® Acrobat® SDK

June 2008

Version 9.0

Adobe® Acrobat® 9.0 SDK Adobe Supplement to the ISO 32000

Edition 1.0, June 2008

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

Preface

The Portable Document Format (PDF) is a file format for representing documents in a manner independent of the application software, hardware, and operating system used to create them and of the output device on which they are to be displayed or printed.

What's in this guide?

This document describes Adobe's extension and implementation notes relative to the soon-to-be-published ISO 32000, *Document management, Portable document format, PDF 1.7*. Until ISO 32000 is published, this document will reference the PDF specification represented by the *PDF Reference, sixth edition, version 1.7* (Nov. 2006) and the *PDF Redaction: Addendum to the PDF Reference, sixth edition, version 1.7*. The next version of this document to be released after ISO 32000 is published will reference specifications in ISO 32000.

The extensions are to the PDF document format. Adobe plans to submit these extensions to ISO as candidates for inclusion into the next version of the ISO 32000 specification. The implementation notes are differences between the PDF specification, including the extensions, and what is implemented in Adobe PDF applications, such as Adobe Acrobat.

Who should read this guide?

This guide is intended for developers of applications that consume or produce PDF content. The implementation notes are intended for developers who need more detail about the Adobe Acrobat family of products.

Related documentation

The resources in this table can help you understand the material in this document. These documents, with the exception of *ISO 32000*, are available through the Adobe PDF Technology Center (select "PDF Specification, Sixth Edition").

For information about	Resource
PDF specification	*PDF Reference, sixth edition, version 1.7*
PDF specification for redaction annotations	*PDF Redaction: Addendum to the PDF Reference, sixth edition, version 1.7*
Corrections to the PDF specification	*Errata for the PDF Reference, sixth edition, version 1.7*
Changes made to content taken from Adobe PDF 1.7 Reference and ported into the ISO32000 draft.	*ISO 32000 - Summary of Changes*, *About the ISO Draft of the PDF 1.7 Reference*
ISO version of the PDF Reference. This document is not yet publicly available.	*ISO 32000*

Transitioning the PDF Specification to ISO

Adobe has transferred responsibility for the PDF specification to the International Standards Organization (ISO). As of this document's publication date, ISO 32000, *Document management, Portable document format, PDF 1.7* has not been published. Consequently, this document references the last PDF specifications published by Adobe, which are the *PDF Reference, sixth edition, version 1.7* (Nov. 2006), the *PDF Redaction: Addendum to the PDF Reference, sixth edition, version 1.7*, and the *Errata for the PDF Reference, sixth edition, version 1.7*.

Extensions to the PDF specification

The language extensions described in this guide may be submitted to ISO as proposed changes to the PDF specification. ISO may or may not accept these Adobe extensions. If accepted, the exact syntax and semantics of the ISO version of these extensions may differ from that described in this guide.

In the past, Adobe identified extensions to the PDF specification with PDF version identifiers, such as PDF 1.7. Now that ISO has responsibility for the PDF specification, such identification is no longer within Adobe's control. As an alternative, Adobe will use the extensions mechanisms defined for ISO 32000-1 by which Adobe and other companies can identify extensions to the PDF specification.

The new convention lets companies and other entities identify their own extensions relative to a base version of PDF. Additionally, the convention identifies extension levels relative to that base version. For normative information on the use of base versions and extension levels in a PDF document, see "3.6.1 Document Catalog" on page 23.

Behavior of PDF consumers that does not support an extension

The extension convention also relies on the behavior of PDF consuming applications when they encounter PDF properties they do not recognize. The Section 2.2.8 "Extensibility" in the *PDF Reference, sixth edition, version 1.7* describes this behavior as follows:

> PDF is designed to be extensible. Not only can new features be added, but applications based on earlier versions of PDF can behave reasonably when they encounter newer features that they do not understand. Appendix H describes how a PDF consumer application should behave in such cases.

And Appendix H in *PDF Reference, sixth edition, version 1.7* provides this additional guidance:

> Both viewer applications and PDF have been designed to enable users to view everything in the document that the viewer application understands and to ignore or inform the user about objects not understood. The decision whether to ignore or inform the user is made on a feature-by-feature basis.

Subsequent extensions to a BaseVersion

Each subsequent company-specific extension level for a particular PDF base version is cumulative. In other words, PDF-processing applications that support the third extension level for a particular base level shall also support the first and second extension levels.

It is Adobe's intent to increment the base version extensions only with major releases of its PDF-processing applications.

Plans related to the first version of ISO 32000

Adobe expects that the first version of ISO 32000 will convey the same requirements as the *PDF Reference, sixth edition, version 1.7*. Because of this similarity and to avoid unnecessary confusion, Adobe will continue using 1.7 as its base version until the second release of ISO 32000.

For information about changes Adobe made to *PDF Reference, sixth edition, version 1.7* to become the ISO Draft International Standard (DIS), see *ISO 32000 — Summary of Changes*. This document is available through www.adobe.com/go/pdf_developer (select "PDF Specification, Sixth Edition").

Plans related to subsequent versions of ISO 32000

After the second version of ISO 32000 is released, Adobe will use a new base version value that reflects the new Version value specified by that second version of ISO 32000. Features that are included in the second ISO version will henceforth be considered as features of the ISO version and the designation (in the PDF files) as being Adobe extensions will no longer be used.

Extensions described for the 1.7 base version that are incorporated into the second version of ISO 32000 will not be included in future versions of this document. Extensions described in a prior release that are not incorporated into ISO 32000 will remain in this document with their original base version and extension level.

The third and subsequent releases of ISO 32000 will repeat this pattern.

Part I: Extensions to the PDF Specification

What's New

This section provides an overview of the extensions to the PDF specification.

Adobe BaseVersion 1.7 and ExtensionLevel 3

The Adobe PDF extensions for BaseVersion 1.7 and Adobe ExtensionLevel 3 extend the PDF specification with the features introduced here. Each feature description references the format changes that make the feature possible.

Extension identification

Extension identification lets companies and other entities identify their own extensions relative to a base version of PDF. Additionally, the convention identifies extension levels relative to that base version. (See "TABLE 3.25 Entries in the catalog dictionary" on page 23.)

Bates numbering page artifact

The Bates number page artifact lets PDF processors override the starting point for the Bates Numbering for the pages in a PDF document. This feature ensures consistent Bates Numbering for pages that were extracted from another PDF document. (See "TABLE 10.17 Property list entries for artifacts" on page 100.)

Features for Architecture, Engineering and Construction

Enforced viewer preferences

Enforced viewer preferences instruct a conforming reader that the user cannot override viewer settings specified in PDF documents. Examples of such viewer preferences are default print scaling value and number of copies.

In one application of this feature, consider a PDF document that contains architectural or engineering drawings. If the scaling of such a document were changed for printing, the document may be compromised. This feature can be used to enforce default print scaling, which ensures that the document is printed with the default scaling. (See "Table 8.1 Entries in a viewer preferences dictionary" on page 28.)

Persistence of 3D measurements and markup

3D measurement and markup data can now be stored in the PDF file, facilitating the communication of 3D shape information between users. The introduction of 3D annotations to the PDF format lets users create views of 3D data to illustrate important 3D features. The addition of 3D measurement definitions also lets users more accurately describe 3D data. This capability is a key step in using a PDF document as the primary mechanism for transferring information in manufacturing and AEC workflows. (See "9.5.6 Persistence of 3D Measurements and Markups" on page 59.)

Accessibility

A PDF document can specify the order in which user tabs cause attention to move from one annotation to the next. (See "TABLE 3.27 Entries in a page object" on page 23.)

Portable collections

The following are changes to the PDF language in support of portable collections that use a SWF file-based presentation of the collection contents:

- New entries in the collection dictionary support SWF file-driven layouts of a collection. (See Table 8.6 on page 29.) The SWF file that manages the presentation is called a *navigator*. The `Navigator` entry is an indirect reference to a navigator dictionary, which contains information about the layout to be used to present the collection. For information about the navigator dictionary, see "Navigators" on page 34.
- Users can decouple the splitter orientation from the `View` key in a collection dictionary. This enhancement provides a details view or tile view with either a horizontal or vertical split. (See Table 8.6 of Section 8.2.4 on page 29.)
- Introduction of a `Thumb` entry to the file specification dictionary. (See Table 3.41 on page 26.) The entry is used to create thumbnail images in conjunction with portable collections that use navigators.
- The `CompressedSize` value of the `Subtype` key in the collection field dictionary is used to display the compressed file sizes for embedded files. (See Table 8.8 on page 30.)
- The new `Colors` entry in a collection dictionary (Table 8.6 on page 29) takes as its value a collection colors dictionary (Table 8.6a on page 31), which is used to specify a list of colors that a navigator should use in its layout of a portable collection.

Rich media

There is a new annotation subtype of `RichMedia`. (See Section 8.4.5 "Annotation Types" on page 38.) The rich media annotation means that Flash applications, video, audio, and other multimedia can be attached to a PDF document. The rich media annotation incorporates the existing 3D annotation structure to support multiple multimedia file assets, including Flash video and compatible variations on the H.264 format. The new constructs allow a two-way scripting bridge between the Flash player and Acrobat. There is support for generalized linking of a Flash application state to a comment or view, which enables video commenting. Finally, actions can be linked to video chapter points. (See the section on "Rich Media" beginning on page 76.)

Seed values and locking a document for digital signatures

Seed values specify constraining information that are used at the time a digital signature is applied. Acrobat 9.0 has two new seed values. `LockDocument` supports controlling the user interface for locking documents after the signature is applied. `AppearanceFilter` filters signature appearances that can be used when signing or certifying a document by name. Additionally, a new entry in the signature field lock dictionary supports the locking after signature feature. (See "Signature Fields" on page 43.)

Encryption and passwords

For Acrobat 9.0, encryption of data uses the AES-256 algorithms. In addition, a new password algorithm supports the use of Unicode-based passwords and passphrases.

There are three parts to the new encryption and password design:

- A new AES variant (`AESV3`) uses AES-256 encryption.
- A new Crypt Filter Version code (5) enables 256-bit (32-byte) keys and does not try to mix the Object ID with the encryption key.
- A new Crypt Revision value (5) activates a new Password algorithm.

See "3.5 Encryption" on page 13.

Barcode form fields

The PDF language additions that support barcode form fields are now documented for Adobe extension level 3. The barcodes feature adds an entry to the widget annotation dictionary. (See Table 8.39 on page 39.) This entry takes as its value a *PaperMetaData generation parameters dictionary.* (See Table 8.39b on page 46.) It also adds an entry to the form field dictionary. (See Table 8.39a on page 46.) For more information about barcode form fields, see the "Barcode Fields" on page 45.

PDF/A-2 and XML form data

A new entry, `XFAResources`, is added to the names dictionary. (See Table 3.28 on page 23.) This entry can be used to save XML form data within a PDF/A-2 conforming file. (See "Incorporation of XFA Datasets into a PDF/A-2 Conforming File" on page 48.)

Adobe BaseVersion 1.7 and ExtensionLevel 1

Acrobat 8.1 extended the PDF language for several 3D-related features. These extensions occurred before the BaseVersion and ExtensionLevel properties were established. The Acrobat 8.1 PDF extensions were finalized in April 2007 and are included in the Adobe PDF extensions for Adobe BaseVersion 1.7 and ExtensionLevel 1.

These extensions were originally specified in *Acrobat Implementation of the PDF Specification*, which is available at the Adobe PDF Technology Center. Select the PDF Specification link.

PRC in 3D annotations

Starting with Acrobat 8.1, 3D annotations can specify streams that conform to the PRC file format. Acrobat 8.1 introduced extended existing PDF features to add support for PRC and to extend support for U3D. PRC is a new highly-compressed CAD visualization format. U3D is another standard for 3D representations. (See TABLE 9.39 Entries in a 3D view dictionary on page 57, and TABLE 9.47 Entries in a 3D node dictionary on page 58.)

Other

Universal 3D file format

PDF 1.6 and later provide support for ECMA-363, Universal 3D file format, edition 1. Acrobat 8.1 extends this support to ECMA-363, Universal 3D file format, 3rd Edition. This feature affects the bibliography. (See "Bibliography" on page 102.)

Support for rich text conventions

Acrobat 8.1 extends support for the rich text conventions described in XML Forms Architecture (XFA) versions 2.5 and 2.6. (See *"Table 8.73 Attributes of the <body> element" on page 42.)*

Syntax (Chapter 3 in PDF Reference)

This section describes extensions to the PDF specification contained in the *PDF Reference, sixth edition, version 1.7* and in ISO 32000 (*Document management - Portable document format - PDF 1.7*). The latter document is the version of the PDF specification that has been ratified by the ISO. These documents differ as described in *About the ISO Draft of the PDF 1.7 Reference*.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

3.5 Encryption

On page 116, modify the paragraph as shown below where unchanged content is shown in gray.

The `V` entry, in specifying which algorithm to use, determines the length of the encryption key, on which the encryption (and decryption) of data in a PDF file is based. For `V` values 2 and 3, the `Length` entry specifies the exact length of the encryption key. In PDF 1.5, a value of 4 for V permits the security handler to use its own encryption and decryption algorithms and to specify *crypt filters* with a key length of 128 bits (16 bytes) to use on specific streams (see Section 3.5.4, "Crypt Filters"). For extension level 3, a value of `5` for `V` permits the specification of crypt filters with a key length of 256 bits (32 bytes).

Modify Table 3.18 as shown below where only entries that changed are listed in the table. The V entry has a new value of 5; the only change to the other listed entries (StmF, StrF, and EFF) is to qualify them for use when V is 5. Unchanged text is shown in gray.

TABLE 3.18 Entries common to all encryption dictionaries

KEY	TYPE	DESCRIPTION
V	number	(*Optional but strongly recommended*) A code specifying the algorithm to be used in encrypting and decrypting the document: 0, An algorithm that is undocumented and no longer supported, and whose use is strongly discouraged. Old files may use this value but no new files should be written using 0. 1, Algorithm 3.1 on page 119, with an encryption key length of 40 bits; see below. 2, (PDF 1.4) Algorithm 3.1, but permitting encryption key lengths greater than 40 bits. 3, (PDF 1.4) An unpublished algorithm that permits encryption key lengths ranging from 40 to 128 bits; see implementation note 22 in Appendix H. 4, (PDF 1.5) The security handler defines the use of encryption and decryption in the document, using the rules specified by the CF, StmF, and StrF entries using algorithm 3.1 with a key length of 128 bits. 5, (*ExtensionLevel 3*) The security handler defines the use of encryption and decryption in the document, using the rules specified by the CF, StmF, and StrF entries using algorithm 3.1a with a key length of 256 bits. The default value if this entry is omitted is 0, but a value of 1 or greater is strongly recommended. (See implementation note 23 in Appendix H.)
StmF	name	(*Optional; meaningful only when the value of V is 4, or 5 for ExtensionLevel 3*) The name of the crypt filter that is used by default when decrypting streams. The name must be a key in the CF dictionary or a standard crypt filter name specified in Table 3.23. All streams in the document, except for cross-reference streams (see Section 3.4.7, "Cross-Reference Streams") or streams that have a Crypt entry in their Filter array (see Table 3.5), are decrypted by the security handler, using this crypt filter. Default value: Identity.

TABLE 3.18 Entries common to all encryption dictionaries

KEY	TYPE	DESCRIPTION
StrF	name	(*Optional; meaningful only when the value of* V *is* 4, *or* 5 *for ExtensionLevel 3*) The name of the crypt filter that is used when decrypting all strings in the document. The name must be a key in the CF dictionary or a standard crypt filter name specified in Table 3.23. Default value: Identity.
EFF	name	(*Optional; meaningful only when the value of* V *is* 4, *or 5 for ExtensionLevel 3*) The name of the crypt filter that should be used by default when encrypting embedded file streams; it must correspond to a key in the CF dictionary or a standard crypt filter name specified in Table 3.23. This entry is provided by the security handler. (See implementation note 24 in Appendix H.) Applications should respect this value when encrypting embedded files, except for embedded file streams that have their own crypt filter specifier. If this entry is not present, and the embedded file stream does not contain a crypt filter specifier, the stream should be encrypted using the default stream crypt filter specified by StmF

3.5.1 General Encryption Algorithm

On page 119, modify the paragraph that begins with "PDF's standard encryption methods also make use…" as well as the paragraph that follows as shown below. Unchanged content is shown in gray. One sentence is struck out and replaced by another to include Encrypt version 5, introduced in extension level 3.

For Encrypt versions 1-4, PDF's standard encryption methods also make use of the MD5 message-digest algorithm for key generation purposes (described in Internet RFC 1321, The MD5 Message-Digest Algorithm; see the Bibliography). Encrypt version 5 does not use MD5.

The encryption of data in a PDF file is based on the use of an encryption key computed by the security handler. Different security handlers compute the encryption key using their own mechanisms. ~~Regardless of how the key is computed, its use in the encryption of data is always the same (see Algorithm 3.1). Because the RC4 algorithm and AES algorithms are symmetric, this same sequence of steps can be used both to encrypt and to decrypt data.~~ Encryption of data uses one of two algorithms. For Encrypt versions 1-4 (through PDF version 1.7), algorithm 3.1 is used. For Encrypt version 5 (extension level 3), algorithm 3.1a is used. The difference is that algorithm 3.1a uses the starting key directly and does not modify the key at all. Algorithm 3.1a is used only with the AES algorithm and 256-bit keys.

Following Algorithm 3.1, insert Algorithm 3.1a, to be used when the value of the V *key is* 5.

Algorithm 3.1a *Encryption of data using the AES algorithm*

1. Use the 32-byte file encryption key for the AES-256 symmetric key algorithm, along with the string or stream data to be encrypted.

   Use the AES algorithm in Cipher Block Chaining (CBC) mode, which requires an initialization vector. The block size parameter is set to 16 bytes, and the initialization vector is a 16-byte random number that is stored as the first 16 bytes of the encrypted stream or string.

   The output is the encrypted data to be stored in the PDF file.

3.5.2 Standard Security Handler

Revise the first paragraph on page 122 to read as follows.

If ~~revision~~ version 4 or 5 is specified, the standard security handler supports crypt filters (see Section 3.5.4, "Crypt Filters"). The support is limited to the `Identity` crypt filter (see Table 3.23) and crypt filters named `StdCF` whose dictionaries contain ~~a `CFM` value of `V2` or `AESV2` and~~ an `AuthEvent` value of `DocOpen`. For version 4, the filter `CFM` value may be `V2` (RC4) or `AESV2` (AES-128). For version 5, the filter `CFM` value shall be `AESV3` (AES-256).

Standard Encryption Dictionary

Table 3.19 shows the encryption dictionary entries for the standard security handler (in addition to those in Table 3.18).

Add three new entries, `OE`, `UE` and `Perms`, to Table 3.19, and modify the other entries as shown. Unchanged content is shown in gray.

TABLE 3.19 Additional encryption dictionary entries for the standard security handler

KEY	TYPE	VALUE
`R`	number	(*Required*) A number specifying which revision of the standard security handler should be used to interpret this dictionary: • `2` if the document is encrypted with a `V` value less than `2` (see Table 3.18) and does not have any of the access permissions set (by means of the P entry, below) that are designated "Revision 3 or greater" in Table 3.20 • `3` if the document is encrypted with a `V` value of `2` or `3`, or has any "Revision 3 or greater" access permissions set • `4` if the document is encrypted with a `V` value of 4 • `5` (*ExtensionLevel 3*) if the document is encrypted with a `V` value of `5`

TABLE 3.19 Additional encryption dictionary entries for the standard security handler

KEY	TYPE	VALUE
O	string	(*Required*) A string used in computing the encryption key. The value of the string depends on the value of the revision number, the R entry described above. **The value of R is 4 or less:** A 32-byte string, based on both the owner and user passwords, that is used in computing the encryption key and in determining whether a valid owner password was entered. **The value for R is 5:** (*ExtensionLevel 3*) A 48-byte string, based on the owner and user passwords, that is used in computing the encryption key and in determining whether a valid owner password was entered. For more information, see "Encryption Key Algorithm" on page 124 and "Password Algorithms" on page 126.
U	string	(*Required*) A string based on the user password. The value of the string depends on the value of the revision number, the R entry described above. **The value of R is 4 or less:** A 32-byte string, based on the user password, that is used in determining whether to prompt the user for a password and, if so, whether a valid user or owner password was entered. **The value for R is 5:** (*ExtensionLevel 3*) A 48-byte string, based on the user password, that is used in determining whether to prompt the user for a password and, if so, whether a valid user password was entered. For more information, see "Password Algorithms" on page 126.
OE	string	(*ExtensionLevel 3; required if R is 5*) A 32-byte string, based on the owner and user passwords, that is used in computing the encryption key. For more information, see "Password Algorithms" on page 126.
UE	string	(*ExtensionLevel 3; required if R is 5*) A 32-byte string, based on the user password, that is used in computing the encryption key. For more information, see "Password Algorithms" on page 126.
P	integer	(*Required*) A set of flags specifying which operations are permitted when the document is opened with user access (see Table 3.20).

TABLE 3.19 Additional encryption dictionary entries for the standard security handler

KEY	TYPE	VALUE
Perms	string	(*ExtensionLevel 3; required if R is 5*) A16-byte string, encrypted with the file encryption key, that contains an encrypted copy of the permission flags. For more information, see "Password Algorithms" on page 126.
EncryptMetadata	boolean	(*Optional; meaningful only when the value of V is 4 or 5; PDF 1.5*) Indicates whether the document-level metadata stream (see Section 10.2.2, "Metadata Streams") is to be encrypted. Applications should respect this value. Default value: true.

Encryption Key Algorithm

Modify the first paragraph of this section as shown.

As noted earlier, one function of a security handler is to generate an encryption key for use in encrypting and decrypting the contents of a document. Given a password string, the standard security handler computes an encryption key. For revision 4 and earlier, the algorithm is as shown in Algorithm 3.2. For revision 5, the algorithm is as shown in Algorithm 3.2a.

Algorithm 3.2 *Computing an encryption key*

1. The password string is generated from OS codepage characters by first converting the string to PDFDocEncoding. If the input is Unicode, first convert to a codepage encoding, and then to PDFDocEncoding for backward compatibility.
2. Initialize the MD5 hash function and pass the result of step 1 as input to this function.
3. Pass the value of the encryption dictionary's O entry to the MD5 hash function. (Algorithm 3.3 shows how the O value is computed.)
4. Treat the value of the P entry as an unsigned 4-byte integer and pass these bytes to the MD5 hash function, low-order byte first.
5. Pass the first element of the file's file identifier array (the value of the ID entry in the document's trailer dictionary; see Table 3.13 on page 97) to the MD5 hash function. (See implementation note 26 in Appendix H.)
6. (*Revision 4 or greater*) If document metadata is not being encrypted, pass 4 bytes with the value 0xFFFFFFFF to the MD5 hash function.
7. Finish the hash.
8. (*Revision 3 or greater*) Do the following 50 times: Take the output from the previous MD5 hash and pass the first n bytes of the output as input into a new MD5 hash, where n is the number of bytes of the encryption key as defined by the value of the encryption dictionary's Length entry.

9. Set the encryption key to the first n bytes of the output from the final MD5 hash, where n is always 5 for revision 2 but, for revision 3 or greater, depends on the value of the encryption dictionary's `Length` entry.

This algorithm, when applied to the user password string, produces the encryption key used to encrypt or decrypt string and stream data according to Algorithm 3.1 on page 119. Parts of this algorithm are also used in the algorithms described below.

Insert Algorithm 3.2a as shown below.

Algorithm 3.2a *Computing an encryption key*

To understand the algorithm below, it is necessary to treat the `O` and `U` strings in the `Encrypt` dictionary as made up of three sections. The first 32 bytes are a hash value (explained below). The next 8 bytes are called the Validation Salt. The final 8 bytes are called the Key Salt.

1. The password string is generated from Unicode input by processing the input string with the SASLprep (IETF RFC 4013) profile of stringprep (IETF RFC 3454), and then converting to a UTF-8 representation.

2. Truncate the UTF-8 representation to 127 bytes if it is longer than 127 bytes.

3. Test the password against the owner key by computing the SHA-256 hash of the UTF-8 password concatenated with the 8 bytes of owner Validation Salt, concatenated with the 48-byte `U` string. If the 32-byte result matches the first 32 bytes of the `O` string, this is the owner password.

   Compute an intermediate owner key by computing the SHA-256 hash of the UTF-8 password concatenated with the 8 bytes of owner Key Salt, concatenated with the 48-byte `U` string. The 32-byte result is the key used to decrypt the 32-byte `OE` string using AES-256 in CBC mode with no padding and an initialization vector of zero. The 32-byte result is the *file encryption key*.

4. Test the password against the user key by computing the SHA-256 hash of the UTF-8 password concatenated with the 8 bytes of user Validation Salt. If the 32 byte result matches the first 32 bytes of the `U` string, this is the user password.

   Compute an intermediate user key by computing the SHA-256 hash of the UTF-8 password concatenated with the 8 bytes of user Key Salt. The 32-byte result is the key used to decrypt the 32-byte `UE` string using AES-256 in CBC mode with no padding and an initialization vector of zero. The 32-byte result is the file encryption key.

5. Decrypt the 16-byte `Perms` string using AES-256 in ECB mode with an initialization vector of zero and the file encryption key as the key. Verify that bytes 9-11 of the result are the characters 'a', 'd', 'b'. Bytes 0-3 of the decrypted `Perms` entry, treated as a little-endian integer, are the user permissions. They should match the value in the `P` key.

Password Algorithms

Revise the opening paragraphs of this section as indicated below.

In addition to the encryption key, the standard security handler must provide the contents of the encryption dictionary (Table 3.18 on page 116 and Table 3.19 on page 122). The values of the `Filter`, `V`, `Length`, `R`, and `P` entries are straightforward. ~~but the computation of the O (owner password) and U (user password) entries requires further explanation. Algorithms 3.3 through 3.5 show how the values of the owner password and user password entries are computed (with separate versions of the latter depending on the revision of the security handler).~~The computation of the values for the `O` (owner password) and `U`

(user password) entries for encryption revision 4 and earlier, and the O, U, OE (owner encryption key), UE (user encryption key) and Perms (permissions) values for encryption revision 5 require more explanation.

Algorithms 3.3 through 3.5 show how the values of the owner password and user password are computed for revision 4 and earlier. Algorithms 3.6 and 3.7 show how to determine if a password is valid. Algorithms 3.8 through 3.10 show how the values for revision 5 are computed. Algorithms 3.11 through 3.13 show how to determine if a revision 5 password is valid.

Passwords for revision 4 and earlier are up to 32 characters in length, and are limited to characters in the PDFDocEncoding character set (see Appendix D).

Following the above paragraphs comes the listing of algorithms 3.3–3.7. These algorithms and accompanying text remain unchanged. Then insert the following paragraphs.

In revision 4 and earlier, the result of running the password algorithm was exactly the file encryption key. In revision 5, the file encryption key is decoupled from the password algorithm to make the owner and user keys independent. For the algorithms below, first generate a 256-bit (32 byte) encryption key for the file using a strong random number generator.

All passwords for revision 5 are based on Unicode. Preprocessing of a user-entered password consists first of normalizing its representation by applying the "SASLPrep" profile (see RFC 4013) of the "stringprep" algorithm (see RFC 3454) to the supplied password using the Normalize and BIDI options. Next, convert the password string to UTF-8 encoding, and then truncate to the first 127 bytes if the string is longer than 127 bytes.

Algorithm 3.8 *Computing the encryption dictionary's U (user password) and UE (user encryption key) values*

1. Generate 16 random bytes of data using a strong random number generator. The first 8 bytes are the User Validation Salt. The second 8 bytes are the User Key Salt. Compute the 32-byte SHA-256 hash of the password concatenated with the User Validation Salt. The 48-byte string consisting of the 32-byte hash followed by the User Validation Salt followed by the User Key Salt is stored as the U key.

2. Compute the 32-byte SHA-256 hash of the password concatenated with the User Key Salt. Using this hash as the key, encrypt the file encryption key using AES-256 in CBC mode with no padding and an initialization vector of zero. The resulting 32-byte string is stored as the UE key.

Algorithm 3.9 *Computing the encryption dictionary's O (owner password) and OE (owner encryption key) values*

1. Generate 16 random bytes of data using a strong random number generator. The first 8 bytes are the Owner Validation Salt. The second 8 bytes are the Owner Key Salt. Compute the 32-byte SHA-256 hash of the password concatenated with the Owner Validation Salt and then concatenated with the 48-byte U string as generated in Algorithm 3.8. The 48-byte string consisting of the 32-byte hash followed by the Owner Validation Salt followed by the Owner Key Salt is stored as the O key.

2. Compute the 32-byte SHA-256 hash of the password concatenated with the Owner Key Salt and then concatenated with the 48-byte U string as generated in Algorithm 3.8. Using this hash as the key, encrypt the file encryption key using AES-256 in CBC mode with no padding and an initialization vector of zero. The resulting 32-byte string is stored as the OE key.

Algorithm 3.10 *Computing the encryption dictionary's Perms (permissions) value*

Fill a 16-byte block as follows:

1. Extend the permissions (contents of the `P` integer) to 64 bits by setting the upper 32 bits to all 1's. (This allows for future extension without changing the format.)

2. Record the 8 bytes of permission in the bytes 0-7 of the block, low order byte first.

3. Set byte 8 to the ASCII value 'T' or 'F' according to the `EncryptMetadata` Boolean.

4. Set bytes 9-11 to the ASCII characters 'a', 'd', 'b'.

5. Set bytes 12-15 to 4 bytes of random data, which will be ignored.

6. Encrypt the 16-byte block using AES-256 in ECB mode with an initialization vector of zero, using the file encryption key as the key. The result (16 bytes) is stored as the `Perms` string, and checked for validity when the file is opened.

Algorithm 3.11 *Authenticating the User Password*

1. Test the password against the user key by computing the SHA-256 hash of the UTF-8 password concatenated with the 8 bytes of User Validation Salt. If the 32-byte result matches the first 32 bytes of the `U` string, this is the user password.

Algorithm 3.12 *Authenticating the Owner Password*

1. Test the password against the user key by computing the SHA-256 hash of the UTF-8 password concatenated with the 8 bytes of Owner Validation Salt and the 48 byte `U` string. If the 32 byte result matches the first 32 bytes of the `O` string, this is the user password.

Algorithm 3.13 *Validating the Permissions*

1. Decrypt the 16 byte `Perms` string using AES-256 in ECB mode with an initialization vector of zero and the file encryption key as the key. Verify that bytes 9-11 of the result are the characters 'a', 'd', 'b'. Bytes 0-3 of the decrypted `Perms` entry, treated as a little-endian integer, are the user permissions. They should match the value in the `P` key. Byte 8 should match the boolean value of the `EncryptMetadata` key.

3.5.3 Public-Key Security Handlers

Public-Key Encryption Algorithms

On page 131, modify the paragraph shown below by inserting the indicated phrase.

The encryption key that is used by Algorithm 3.1 is calculated by means of an a SHA-1 message digest operation for a key length of 128 bits or a SHA-256 digest operation for a key length of 256 bits that digests the following data, in order:

3.5.4 Crypt Filters

Modify Table 3.22 where only changed entries are shown. Unchanged content is shown in gray.

TABLE 3.22 Entries common to all crypt filter dictionaries

KEY	TYPE	VALUE
CFM	name	(Optional) The method used, if any, by the consumer application to decrypt data. The following values are supported: None The application does not decrypt data but directs the input stream to the security handler for decryption. (See implementation note 30 in Appendix H.) V2 The application asks the security handler for the encryption key and implicitly decrypts data with Algorithm 3.1, using the RC4 algorithm. AESV2 (*PDF 1.6*) The application asks the security handler for the encryption key and implicitly decrypts data with Algorithm 3.1, using the AES-128 algorithm in Cipher Block Chaining (CBC) with padding mode with a 16-byte block size and an initialization vector that is randomly generated and placed as the first 16 bytes in the stream or string. The key size (Length) shall be 128 bits. AESV3 (*ExtensionLevel 3*) The application asks the security handler for the encryption key and implicitly decrypts data with Algorithm 3.1a, using the AES-256 algorithm in Cipher Block Chaining (CBC) with padding mode with a 16-byte block size and an initialization vector that is randomly generated and placed as the first 16 bytes in the stream or string. The key size (Length) shall be 256 bits. When the value is V2, AESV2, or AESV3, the application may ask once for this encryption key and cache the key for subsequent use for streams that use the same crypt filter. Therefore, there must be a one-to-one relationship between a crypt filter name and the corresponding encryption key. Only the values listed here are supported. Applications that encounter other values should report that the file is encrypted with an unsupported algorithm. Default value: None.
Length	integer	(*Required if the value for the V key in Table 3.18 is 2 or 3, optional otherwise*) The bit length of the encryption key. It must be a multiple of 8 in the range of 40 to ~~128~~ 256. **Note:** Security handlers can define their own use of the Length entry but are encouraged to use it to define the bit length of the encryption key.

3.6 Document Structure

3.6.1 Document Catalog

Add the following new entry after the `Version` *entry in Table 3.25*

TABLE 3.25 Entries in the catalog dictionary

KEY	SUBTYPE	DESCRIPTION
`Extensions`	dictionary	(*ExtensionLevel 3*) An extensions dictionary representing information about the PDF extensions that this document may contain. Table 3.25a on page 24 describes the entries in the extensions dictionary.

3.6.2 Page Tree

Add new values to the value for the `Tabs` *key in Table 3.27. Unchanged text is shown in gray.*

TABLE 3.27 Entries in a page object

KEY	TYPE	VALUE
`Tabs`	name	*(Optional; PDF 1.5)* A name specifying the tab order that shall be used for annotations on the page. The possible values shall be `R` (row order), `C` (column order), and `S` (structure order). Beginning with ExtensionLevel 3, the possible values also include `A` (annotations array order) and `W` (widget order). See Section 8.4, "Annotations" for details. Regarding annotations array order, see implementation note E-17, page 132.

3.6.3 Name Dictionary

Add the new entry `XFAResources` *to Table 3.28.*

TABLE 3.28 Entries in the name dictionary

KEY	TYPE	VALUE
`XFAResources`	dictionary	(*Optional; ExtensionLevel 3*) A name tree mapping name strings to streams that contain the `<xfa:datasets>` element of the XFA. (See "Incorporation of XFA Datasets into a PDF/A-2 Conforming File" on page 48.)

3.6.4 Extensions to PDF

Add this new section (including the above title) after Section 3.6.3 "Name Dictionary."

Beginning with BaseVersion 1.7, the extensions dictionary lets developers designate that a given document contains extensions to PDF. The presence of the extension dictionary in a document indicates

that it may contain developer-specific PDF properties that extend a particular base version of the PDF specification.

The extensions dictionary enables developers to identify their own extensions relative to a base version of PDF. Additionally, the convention identifies extension levels relative to that base version. The intent of this dictionary is to enable developers of PDF-producing applications to identify company-specific specifications (such as this one) that PDF-consuming applications use to interpret the extensions.

The intent of the extensions convention is two-fold:

- Enable developers of PDF-producing applications to identify the use of company-specific extensions they have added to a PDF document and to associate those extensions with their own publicly-available specifications.
- Enable developers of PDF-consuming applications to determine which extensions are present in a PDF document and to associate those extensions with the specifications that describe them.

To avoid collisions over company names and company-specific extension names, ISO provides the prefix name registry. The *prefix registry* is used to designate a 4-character, case-sensitive prefix that identifies a company or other entity. This prefix is used for company-specific version identifiers. Additionally, these prefixes are used to create second-class names of the form ACME_aPropertyName. This registry is available from the Prefix Registry link in the Related Resources panel at the following Web page:

http://www.adobe.com/go/ISO32000Registry

Table 3.25a shows the entries in an extensions dictionary, where each *Prefix name* entry is a registered prefix. The extensions dictionary is the value of the `Extensions` entry in the document catalog. (See Table 3.25 on page 23.) The extensions dictionary is part of ISO 32000 (See Section 7.12, "Extensions Dictionary.")

TABLE 3.25a Entries in the extensions dictionary

KEY	SUBTYPE	DESCRIPTION
`Type`	name	(*Optional*) The type of PDF object that this dictionary describes; shall be `Extensions` for an extensions dictionary.
`Prefix name`	dictionary	A developer extensions dictionary representing the extensions added by a developer. See "Table 3.25b Entries in a developer extensions dictionary" on page 25. The prefix name registry is available at http://www.adobe.com/go/ISO32000Registry Select from the Prefix Registry link in the Related Resources pane. For information about prefix names, see Appendix E of the *PDF Reference, sixth edition, version 1.7*.

Table 3.25b shows the entries in a developer extensions dictionary.

Table 3.25b Entries in a developer extensions dictionary

KEY	SUBTYPE	VALUE
`Type`	name	(*Optional*) The type of PDF object that this dictionary describes; if present, will be `DeveloperExtensions` for a developer extensions dictionary.
`BaseVersion`	name	(*Required*) A name that designates a version of the PDF specification and that is syntactically consistent with the `Version` key in the `Catalog` entry. (See Section 3.6.1.) The name value 1.7 designates PDF version 1.7, as defined in *PDF Reference, sixth edition, version 1.7* available from the Adobe PDF Technology Center. **Note:** The version in a PDF file can be specified in two places. The initial line in the file or in the Catalog. The `BaseVersion` value may differ from the version number on the header line, or as supplied by the `Version` key in the Catalog. This is because it reflects the version of the standard that is extended and not the version of this particular file.
`ExtensionLevel`	integer	(*Required*) An integer that is used in conjunction with the associated `BaseVersion`. The interpretation of this value is defined by the developer. The value of `ExtensionLevel` for a particular `BaseVersion` value shall increase over subsequent extensions. **Note:** Developers can designate lower-order digits to represent sub-levels. For example, the value 100 could represent 1.0, or the value 101 could represent 1.1.

The PDF document that contains the following segment may contain the Adobe-specific extensions relative to PDF version 1.7.

```
%PDF 1.7
<</Type /Catalog
   /Extensions
      <</ADBE
         <</BaseVersion /1.7
            /ExtensionLevel 3
         >>
      >>
   >>
```

The PDF document that contains the following segment may contain ACME extension number 4, which is relative to PDF version 1.7. It may also contain MyCo extension number 1002, which is relative to PDF version 500 (a hypothetical PDF version number).

```
%PDF 1.7
<</Type /Catalog
   /Extensions
      <</ACME
```

```
        << /BaseVersion /1.7 /ExtensionLevel 4>>
        /MyCo
        << /BaseVersion /500 /ExtensionLevel 1002>>
      >>
>>
```

3.10 File Specifications

3.10.2 File Specification Dictionaries

The following entry is added to the file specification dictionary, Table 3.41.

TABLE 3.41 Entries in a file specification dictionary

KEY	TYPE	VALUE
Thumb	stream	(*Optional; ExtensionLevel 3*) A stream object defining the thumbnail image for the file specification. (See Section 8.2.3, "Thumbnail Images.")

The Thumb entry is used primarily in conjunction with files embedded in portable collections that use a navigator. However, it is extensible to any other feature that may require a thumbnail image representation of a file specification. For information about navigators, see "Navigators" on page 34.

Graphics (Chapter 4 in PDF Reference)

This section describes extensions to the PDF specification contained in the *PDF Reference, sixth edition, version 1.7* and in ISO 32000 (*Document management - Portable document format - PDF 1.7*). The latter document is the version of the PDF specification that has been ratified by the ISO. These documents differ as described in *About the ISO Draft of the PDF 1.7 Reference*.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

4.8 Images

4.8.4 Image Dictionaries

Add new entries, `Measure` *and* `PtData`*, to Table 4.39 in support of geospatial content. See Section* 8.8.1 Geospatial Features *on* page 49.

TABLE 4.39 Additional entries specific to an image dictionary

KEY	TYPE	VALUE
`Measure`	dictionary	(*Optional; ExtensionLevel 3*) A measure dictionary (see Table 8.110, page 49) that specifies the scale and units that apply to the image.
`PtData`	dictionary	(*Optional; ExtensionLevel 3*) A point data dictionary (see Table 111d, page 53) that specifies the extended geospatial data that applies to the image.

4.9 Form XObjects

4.9.1 Form Dictionaries

Add new entries, `Measure` *and* `PtData`*, to Table 4.45 in support of geospatial content. See Section* 8.8.1 Geospatial Features *on* page 49.

TABLE 4.45 Additional entries specific to a type 1 form dictionary

KEY	TYPE	VALUE
`Measure`	dictionary	(*Optional; ExtensionLevel 3*) A measure dictionary (see Table 8.110, page 49) that specifies the scale and units that apply to the form.
`PtData`	dictionary	(*Optional; ExtensionLevel 3*) A point data dictionary (see Table 111d, page 53) that specifies the extended geospatial data that applies to the form.

Interactive Features (Chapter 8 in PDF Reference)

This section describes extensions to the PDF specification contained in the *PDF Reference, sixth edition, version 1.7* and in ISO 32000 (*Document management - Portable document format - PDF 1.7*). The latter document is the version of the PDF specification that has been ratified by the ISO. These documents differ as described in *About the ISO Draft of the PDF 1.7 Reference*.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

8.2 Document-Level Navigation

8.1 Viewer Preferences

Add the new `Enforce` *entry in Table 8.1.*

Table 8.1 Entries in a viewer preferences dictionary

KEY	TYPE	VALUE
`Enforce`	array of names	(*Optional; ExtensionLevel 3*) Viewer preference settings that may be enforced by conforming readers and that will not be overridden by subsequent selections in the application user interface. Table 8.1a Names defined for an Enforce array specifies valid names.

After Table 8.1, and before Section 8.2, add the following paragraph and table.

Table 8.1a shows the only valid name that can appear in an `Enforce` array. Future additions to this table should be limited to keys in the viewer preferences dictionary with the following qualities:

- Can be assigned values (default or specified) that cannot be used in a denial-of-service attack
- Have default values that cannot be overridden using the application user interface

Table 8.1a Names defined for an Enforce array

NAME	DETAILS
`PrintScaling`	(*Optional; ExtensionLevel 3*) This name may appear in the `Enforce` array only if the corresponding entry in the viewer preferences dictionary (Table 8.1) specifies a valid value other than `AppDefault`.

8.2.4 Collections

Following the paragraph (in the PDF Reference) that begins with "The file attachments comprising a collection," insert the two paragraphs below.

Beginning with extension level 3, a portable collection can include a SWF file that creates an interactive layout, or presentation, of the collection contents. The SWF file that manages the collection presentation is called a *navigator*. For more information about navigators, see "Navigators" on page 34.

The collection dictionary contains some entries that support navigators. The `Navigator` entry is an indirect reference to a navigator dictionary that contains a reference to the SWF file that describes the layout. The `Resources` entry is an indirect reference to a name tree that can describe the content and display of a header and welcome page of a portable collection. The value of the `Colors` entry is a *collection colors dictionary* that specifies a suggested set of colors for use by a collection navigator. The `Folders` entry is an indirect reference to the root folder of the collection's folder structure.

The `View` entry of Table 8.6 has an additional value of `C`, and unchanged content is shown in gray. Add the entries `Navigator`, `Resources`, `Colors`, `Folders`, and `Split` to Table 8.6 in support of portable collections that use navigators.

TABLE 8.6 Entries in a collection dictionary

KEY	TYPE	VALUE
`View`	name	(Optional) The initial view. The following values are valid: `D` The collection view is presented in details mode, with all information in the Schema dictionary presented in a multi-column format. This mode provides the most information to the user. `T` The collection view is presented in tile mode, with each file in the collection denoted by a small icon and a subset of information from the Schema dictionary. This mode provides top-level information about the file attachments to the user. `H` The collection view is initially hidden, without preventing the user from obtaining a file list via explicit action. `C` (*ExtensionLevel 3*) The collection view is presented by a custom navigator. Default value: `D`
`Navigator`	dictionary	(*Required if the value of `View` is `C`; ExtensionLevel 3*) An indirect reference to the navigator dictionary that describes the navigator that provides the collection view. See "TABLE 8.6d Entries in a navigator dictionary" on page 34.
`Resources`	name tree	(*Optional; ExtensionLevel 3)* An indirect reference to a name tree, mapping name strings to streams for named resources used by SWF file-based aspects of the collection presentation other than navigators. A conforming application should store data required by the welcome page and header in the collection resources name tree. See implementation note E-1, page 130. See the discussion in "Resources name tree" on page 36.

TABLE 8.6 Entries in a collection dictionary

KEY	TYPE	VALUE
Colors	dictionary	(*Optional; ExtensionLevel 3*) A *collection colors dictionary* specifying a suggested set of colors for use by a collection navigator. See "TABLE 8.6a Entries in a collection colors dictionary" on page 31. **Note:** A navigator accesses the colors dictionary through the Acrobat ActionScript API. The ActionScript property INavigatorHost.navigatorColorPalette provides access to a user-specified color theme or palette, a set of colors that should be used by the navigator for its presentation. (See the *Acrobat ActionScript API Reference* in the Bibliography.) There is no requirement, however, that the navigator actually use the colors.
Folders	dictionary	(*Required if the collection has folders; ExtensionLevel 3*) An indirect reference to a folder dictionary that is the single common ancestor of all other folders in a portable collection. See "TABLE 8.6c Entries in a folder dictionary" on page 32.
Split	dictionary	(*Optional; ExtensionLevel 3*) A collection split dictionary that specifies the orientation of the splitter bar. See "TABLE 8.6b Entries in a collection split dictionary" on page 31. If Split is not present, the preferred orientation is determined by the value of the View key. A value of D (or no value) indicates a horizontal orientation, while a value of T indicates a vertical orientation. No splitter is used if the View key has a value of H.

Add a new value of CompressedSize to the Subtype key in Table 8.8. This value is used by portable collections to display the compressed size of embedded files.

TABLE 8.8 Entries in a collection field dictionary

KEY	TYPE	VALUE
Subtype	name	CompressedSize (*Optional; ExtensionLevel 3*) The field data is the length of the embedded filestream, as identified by the Length entry in the embedded filestream dictionary (see Section 3.10.3, "Embedded File Streams"), and the two values shall be identical.

Add the next three tables to the end of Section 8.2.4.

A *collection colors dictionary* lists colors that a navigator should use in its presentation of a collection. This dictionary has the following entries.

TABLE 8.6a Entries in a collection colors dictionary

KEY	TYPE	VALUE
Background	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing a DeviceRGB color used for the navigator background.
CardBackground	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing a DeviceRGB color used for the navigator card background.
CardBorder	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing a DeviceRGB color used for the navigator card border.
PrimaryText	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing a DeviceRGB color used for the primary text in a navigator, such as file names and links.
SecondaryText	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing a DeviceRGB color used for other text in a navigator.

A portable collection can be displayed in a split view, showing a navigator (or file list) in one sub-view and a preview of the currently selected document in the other. A *collection split dictionary* holds information that specifies the initial orientation (horizontal, vertical, or no split) of the splitter control and its position.

TABLE 8.6b Entries in a collection split dictionary

KEY	TYPE	VALUE
Direction	name	(*Optional; ExtensionLevel 3*) The orientation of the splitter bar. The following values are valid: H indicates that the window is split horizontally. V indicates that the window is split vertically. N indicates that the window is not split. The entire window region is dedicated to the file navigation view.
Position	number	(*Optional; ExtensionLevel 3*) The initial position of the splitter bar, specified as a percentage of the available window area. Values should range from 0 to 100. The entry is ignored if Direction is set to N.

Beginning with extension level 3, a portable collection can contain a *Folders object* for the purpose of organizing files into a hierarchical structure. The structure is represented by a tree with a single root folder acting as the common ancestor for all other folders and files in the collection. The single root folder is referenced in the Folders entry of Table 8.6 on page 29.

Table 8.6c describes the entries in a folder dictionary.

TABLE 8.6c Entries in a folder dictionary

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes; if present, shall be Folder for a folder dictionary.
ID	integer	(*Required; ExtensionLevel 3*) A non-negative integer value representing the unique folder identification number. Two folders shall not share the same ID value. The folder ID value appears as part of the name tree key of any file associated with this folder. A detailed description of the association between folder and files can be found after this table.
Name	text string	(*Required; ExtensionLevel 3*) A file name representing the name of the folder. Two sibling folders shall not share the same name following case normalization. **Note:** Descriptions of file name and case normalization follow this table.
Parent	dictionary	(*Required for child folders; ExtensionLevel 3*) An indirect reference to the parent folder of this folder. This entry shall be absent for a root folder.
Child	dictionary	(*Required if the folder has any descendents; ExtensionLevel 3*) An indirect reference to the first child folder of this folder.
Next	dictionary	(*Required for all but the last item at each level; ExtensionLevel 3*) An indirect reference to the next sibling folder at this level. Siblings should be ordered according to the collection Sort key (Table 8.6, page 29) or by folder name if no collection Sort key is present.
CI	dictionary	(*Optional; ExtensionLevel 3*) The *collection item dictionary* (see Table 3.45 of the *PDF Reference*) associated with this folder.
Desc	text string	(*Optional; ExtensionLevel 3*) A text description associated with this folder.
CreationDate	date string	(*Optional; ExtensionLevel 3*) The date the folder was first created.
ModDate	date string	(*Optional; ExtensionLevel 3*) The date of the most recent change to immediate child files or folders of this folder.

TABLE 8.6c Entries in a folder dictionary

KEY	TYPE	VALUE
Thumb	stream	(*Optional; ExtensionLevel 3*) A stream object defining the thumbnail image for the folder See Section 8.2.3, "Thumbnail Images" of the *PDF Reference*.
Free	array	(*Optional; only used by root folder; ExtensionLevel 3*) An array containing ID values that are not currently in use by the folder structure. The array contains zero or more pairs of numbers, a low value followed by a high value. Each pair represents an endpoint-inclusive range of values that are available for use when a new folder is added. Each low value is less than or equal to its corresponding high value.

New values for the ID key shall be obtained by a conforming reader by accessing the Free entry in the root folder. If an ID value is used from the Free entry array, the array is updated.

As previously described, the Name entry is a file name for a folder. A folder, as well as its associated files, have naming restrictions inherited from the *Universal Container Format (UCF)* specification. (See the Bibliography.) Strings that conform to these restrictions are known as *file names*. A valid file name conforms to the following requirements:

- The string shall be a PDF text string.
- The string shall not contain any embedded NULL characters.
- The number of characters in the string shall be between 1 and 255 inclusive.
- The string shall not contain any of the eight special characters: U+002F SOLIDUS (/), U+005C REVERSE SOLIDUS (\), U+003A COLON (:), U+002A ASTERISK (*), U+0022 QUOTATION MARK ("), U+003C LESS-THAN SIGN (<), U+003E GREATER-THAN SIGN (>), and U+007C VERTICAL LINE (|).
- The last character shall not be a U+002E FULL STOP (.).

A conforming reader may choose to support invalid names or not. If not, an appropriate error message shall be provided.

In addition to the restriction on naming folders, as just described, it is further required that two file names in the same folder do not map to the same string following case normalization. Two file names that differ only in case are disallowed within the same folder. See http://www.unicode.org/reports/tr21/tr21-5.html for information on case normalization.

The CI entry (a collection item dictionary) allows user-defined metadata to be associated with a folder, just as it does for embedded files in a collection.

Folders are indirect objects, and relationships between folders in the tree are specified using Parent, Child, and Next keys.

When folders are used, all files in the EmbeddedFiles name tree (see Table 3.28 of the *PDF Reference*) shall be treated as members of the folder structure by a conforming reader. The association between files and folders is accomplished using a special naming convention on the key strings of the name tree. See the *PDF Reference*, Section 3.8.5, "Name Trees," for a discussion of the key strings. If no folder structure is specified, conforming readers show all files in the collection in a flat list, maintaining compatibility with older viewers.

As previously mentioned, files in the EmbeddedFiles name tree are associated with folders by a special naming convention applied to the name tree key strings. Strings that conform to the following rules serve to associate the corresponding file with a folder:

- The name tree keys are PDF text strings.
- The first character, excluding any byte order marker, is U+003C, the LESS-THAN SIGN (<).
- The following characters shall one or more digits (0 to 9) followed by the closing U+003E, the GREATER-THAN SIGN (>)
- The remainder of the string is a file name.

The section of the string enclosed by LESS-THAN SIGN GREATER-THAN SIGN(<>) is interpreted as a numeric value that specifies the ID value of the folder with which the file is associated. The value shall correspond to a folder ID. The section of the string following the folder ID tag represents the file name of the embedded file.

Files in the EmbeddedFiles name tree that do not conform to these rules shall be treated as associated with the root folder.

Navigators

Beginning with extension level 3, a portable collection can include a SWF file-based interactive layout, or presentation, of the collection contents. The SWF file that manages the presentation is called a *navigator*. A conforming interactive reader shall play the SWF file that drives the presentation of the collection, unless some other considerations such as accessibility require a non-SWF file-based presentation. (See implementation note E-2, page 130.)

This section describes how navigator information is stored in a PDF file, but does not describe the SWF file that implements the navigator itself. (See implementation note E-3, page 130.)

When a portable collection is created, the View entry of the collection dictionary (Table 8.6, page 29) is given a value of C to indicate that the collection view is provided by a custom navigator. A Navigator entry in the collection dictionary contains an indirect reference to the navigator dictionary that describes the navigator. Other entries in the collection dictionary may also be present. These include Resources, Colors, Folders, and Split. See Table 8.6 on page 29 for information about these entries.

The navigator dictionary contains a variety of information about the navigator to be used, such as the location of the SWF file that implements the layout of the collection, a unique ID number, load type, and resources. The following table lists the entries in the navigator dictionary; the entries used only for authoring are noted.

TABLE 8.6d Entries in a navigator dictionary

KEY	TYPE	VALUE
SWF	text string	(*Required; ExtensionLevel 3*) The location of the SWF file that implements the layout of the collection, expressed as an entry in the navigator's resources tree. See the Resources entry later in this table.
Name	text string	(*Required; ExtensionLevel 3; authoring only*) The name to be displayed to the user when choosing a navigator in a conforming interactive reader.

TABLE 8.6d Entries in a navigator dictionary

KEY	TYPE	VALUE
Desc	text string	(*Optional; ExtensionLevel 3; authoring only*) The description of the navigator available to the user when choosing a navigator in a conforming interactive reader.
Category	text string	(*Optional; ExtensionLevel 3; authoring only*) The category name that may be used to group navigators.
ID	text string	(*Required; ExtensionLevel 3; authoring only*) A string representing a unique ID for the navigator, expressed as a URI. See implementation note E-18, page 132.
Version	text string	(*Optional; ExtensionLevel 3; authoring only*) A string representing a numerical version number of the form m[.n[.p[.q]]], where, m, n, p, and q are non-negative integers. If not present, n, p, and q default to 0. See implementation note E-19, page 132.
APIVersion	text string	(*Required; ExtensionLevel 3*) A string representing a numerical version number of the form m[.n[.p[.q]]], where, m, n, p, and q are non-negative integers. If not present, n, p, and q default to 0. This number is the version of the navigator API required by the navigator SWF file. If a portable collection is opened in an older viewer that supports a SWF file driven presentation of collections, it does not load and run the navigator SWF file if this version is greater than that supported by the conforming reader.
LoadType	name	(*Optional; ExtensionLevel 3*) The method to use to load the navigator SWF file. The following values are valid: Module The navigator SWF file is loaded as an Adobe© Flex™ 2 module. Default The navigator SWF file is loaded as an ordinary SWF file. If this key is omitted, the navigator SWF file is loaded as an ordinary SWF file. **Note:** For information about modules, see the chapter "Modular applications overview" in the *Flex 2 Developer Guide*. (See the Bibliography.)
Icon	text string	(*Optional; ExtensionLevel 3; authoring only*) The location of an image representing the navigator expressed as an entry in the navigator's resources tree. The resource is not a PDF XObject, but a PNG or JPEG image accepted by the version of the SWF file identified by the navigator APIVersion. See implementation note E-4, page 130.

TABLE 8.6d Entries in a navigator dictionary

KEY	TYPE	VALUE
`Locale`	text string	(*Optional; ExtensionLevel 3*) A string that specifies a locale according to UTS 35, *Unicode Technical Standard #35*. The string is usually a two-character language code, followed by a two-character territory code, separated by an underscore (_). As noted in UTS 35, the canonical form of a locale ID uses an underscore (_) as a separator, but a processor should treat a hyphen (-) as equivalent to an underscore (_).
`Strings`	name tree	(*Optional; ExtensionLevel 3)* An indirect reference to a string table name tree that maps string IDs to localized strings. Both the keys and values of the name tree are text strings. See the discussion in "String table" on page 37.
`InitialFields`	dictionary	(*Optional; ExtensionLevel 3; authoring only*) A *collection schema dictionary* (see Table 8.7 in the *PDF Reference*) that defines the schema fields expected by the navigator. When a navigator is applied to a portable collection and any of these fields do not exist in the collection schema, the fields are added in the order specified. Existing fields are reordered if necessary to conform to the order of the fields specified here. Existing fields not listed in `InitialFields` are moved to the end of the collection schema.
`Resources`	name tree	(*Required; ExtensionLevel 3)* An indirect reference to a name tree, mapping text strings to streams for named resources including the navigator SWF file, the navigator icon, and resources used by the navigator SWF file. See the discussion in "Resources name tree" that follows this table.

Resources name tree

A resources name tree provides access to named *files* used by portable collection-related SWF file components. Collections contain two resources name trees, one for the navigator (the `Resources` entry in the navigator dictionary, Table 8.6d, page 34) and one for all other components (the `Resources` entry in the collection dictionary, Table 8.6, page 29). The resources in the name tree exist in a hierarchical directory structure, matching the abstract container and directory structure defined by the *Uniform Container Format (UCF)*. (See the Bibliography.) The name tree maps relative IRI references to streams. (For information about *Internationalized Resource Identifiers (IRIs)*, see RFC 3986 and RFC3987 in the Bibliography.) The use of parent directory symbol (..) in a relative IRI is disallowed. As specified by UCF, if a relative IRI reference contains an absolute path, which is an IRI that has no schema or authority but begins with a U+002F SOLIDUS (/), the reference is resolved relative to the root directory of the abstract container.

These relative IRI references are used by SWF files that implement components of the collection user interface, including the custom navigator, the welcome page, and the header. Two components of the navigator itself, the navigator SWF file and its icon, are stored in a resources name tree.

The base IRI for references from the navigator SWF file is the navigator SWF file itself. The navigator SWF file IRI is defined by the `SWF` key in the navigator dictionary. Suppose, for example, that this key specifies the navigator SWF file IRI to be MyNavigator.swf and that this SWF file, in turn, refers internally to a style sheet named MyStyles.css and an image named images/MyImage.jpeg. Suppose also that the `Icon` key in

the navigator dictionary refers to MyIcon.png. The base IRI in this case is the root directory of the abstract container. The navigator's resources name tree should then include four resources: MyNavigator.swf, MyIcon.png, MyStyles.css, and images/MyImage.jpeg.

The base IRI for PDF viewer components such as the welcome page and header is the root directory of the abstract container.

A resources name tree differs from the `EmbeddedFiles` name tree (Section 3.10.3 in the *PDF Reference*) in two respects. The keys in the tree are always text strings that represent relative IRIs. The values in the tree are simple streams rather than file specifications. The only information associated with a resource is its name, which is defined by the key.

Storing the navigator resources in a name tree separate from the collection resources allows a navigator to be self-contained. If a user replaces a navigator with another, the navigator resources are replaced as well. The SWF file components embedded in a PDF file use the collection resources. (See the `Resources` entry in Table 8.6. on page 29.) The resources in the collection resources name tree are organized by directory. All the resources needed by the welcome page are named beginning with welcome/, as if these resources were in a directory named welcome. Similarly, header resources begin with header/.

String table

Navigators include some text that is displayed to the user. While the content of PDF documents is generally not localized, navigators include both content and application (the navigator SWF file), and PDF provides a mechanism that enables navigators to specify localized text separate from the application.

The `String` entry in the navigator dictionary (TABLE 8.6d Entries in a navigator dictionary) is an indirect reference to a string table. This string table is a name tree whose keys are string IDs and values are localized text strings. A string ID is itself a text string. Given a string ID and a locale (the `Locale` entry in a navigator dictionary), a localized text string can be found. The Acrobat ActionScript API provides access to the string table. (See the *Acrobat ActionScript API Reference* in the Bibliography.) The method `INavigatorHost.getLocalizedString` returns the value for a given ID or returns the provided default value (a parameter to `getLocalizedString`) if the ID is not in the string table.

8.4 Annotations

Page 605 of the PDF Reference describes the effect of each supported `Tabs` value. After the description for the `S` (structure order) value, add these bulleted entries:

- `A` (annotation array order): All annotations are visited in the order in which they appear in the page `Annots` array. (See Table 3.27**, "**Entries in a page object," and implementation note E-17, page 132.)
- `W` (widgets order): Widget annotations are visited in the order in which they appear in the page `Annots` array, followed by other annotation types in row order. (See Table 3.27 **"**Entries in a page object."**)** For information about row order, see the `R` (row order) entry description on page 605.

For information about the Acrobat implementation of this feature and the effect of the accessibility preference on annotations order, see "Interaction between accessibility preference and annotation tab order," part of implementation note 77, on page 116.

8.4.5 Annotation Types

Add the `Projection` and the `RichMedia` annotation types to Table 8.20

TABLE 8.20 Annotation types

ANNOTATION TYPE	DESCRIPTION	MARKUP?	DISCUSSED IN SECTION
`Projection`	(*ExtensionLevel 3*) Projection annotation	Yes	"Projection Annotations" on page 39
`RichMedia`	(*ExtensionLevel 3*) RichMedia annotation	No	"9.6.1 RichMedia Annotations" on page 76

Markup Annotations

Add a new subtype to the `ExData` entry in Table 8.21. Unchanged content is shown in gray.

TABLE 8.21 Additional entries specific to markup annotations

KEY	TYPE	VALUE
`ExData`	dictionary	(*Optional; PDF 1.7*) An *external data dictionary* specifying data to be associated with the annotation. This dictionary contains the following entries: `Type` (*optional*): If present, must be ExData. `Subtype` (*required*): A name specifying the type of data that the markup annotation is associated with. `Markup3D` (*PDF 1.7*) for a 3D comment. Additional entries in this dictionary are listed in Table 9.48 on page 835. (See also implementation note 96 in Appendix H.) `3DM` (*ExtensionLevel 3)* for a 3D measurement. Additional entries in this dictionary are listed in "TABLE 9.39g Entries in the external data dictionary of a projection annotation" on page 76. `MarkupGeo` (*ExtensionLevel 3*) for geospatial markup. This `Subtype` does not define any additional entries. ~~For each value of `Subtype`, other entries are defined. Table 9.48 on page 835 lists the values that correspond to a subtype of `Markup3D`. (See also implementation note 96 in Appendix H.)~~

Widget Annotations

Add the PMD entry to the end of Table 8.39, "Additional entries specific to a widget annotation" on page 641. The PMD entry is used by barcode fields.

TABLE 8.39 Additional entries specific to a widget annotation

KEY	TYPE	VALUE
PMD	dictionary	(*Required; barcode fields only; ExtensionLevel 3*) The *PaperMetaData generation parameters dictionary*. The entries of this dictionary are instructions to the barcode encoding software on how to generate the barcode image. See Table 8.39b, page 46, for the description of a PaperMetaData generation parameters dictionary. **Note:** PaperMetaData should not be confused with XMP as used in a PDF file. XMP (Extensible Metadata Platform) is an XML format for representing metadata.

Add the following new topic to the end of Section 8.4.5.

Projection Annotations

A *projection annotation* is a markup annotation subtype (Table 8.20, page 38) that has much of the functionality of other markup annotations. However, a projection annotation is only valid within the context of an associated run-time environment, such as an activated 3D model.

The entries of a annotation dictionary for a projection annotation are those listed in Table 8.15 and Table 8.21 of the *PDF Reference*.

Projection annotations provide a way to save 3D and other specialized measurements and comments as markup annotations. These measurements and comments then persist in the document.

When a projection annotation is used in conjunction with a 3D measurement ("3D Measurements and Projection Annotations" on page 75), it has an ExData dictionary with a Subtype of 3DM. (See TABLE 9.39g Entries in the external data dictionary of a projection annotation on page 76.) Otherwise, the ExData dictionary is optional.

A projection annotation with a Rect entry that has zero height or zero width does not have an AP dictionary.

8.5 Actions

8.5.3 Action Types

Add the `rich-media-execute` action to Table 8.48.

TABLE 8.48 Action Types

ACTION TYPE	DESCRIPTION	DISCUSSED IN SECTION
`RichMediaExecute`	(*ExtensionLevel 3; RichMedia annotation only*) Specifies a command to be sent to the annotation's handler.	"Rich-Media-Execute Actions" on page 40

Rich-Media-Execute Actions

A *rich-media-execute action* identifies a rich media annotation and specifies a command to be sent to that annotation's handler. (See Section 9.6, "Rich Media" on page 76.) Table 8.48a shows the entries in a *rich-media-execute action dictionary*.

TABLE 8.48a Additional entries specific to a rich-media-execute action

KEY	TYPE	DESCRIPTION
`S`	name	(*Required; ExtensionLevel 3*) The type of action that this dictionary describes; shall be `RichMediaExecute` for a rich-media-execute action.
`TA`	dictionary	(*Required; ExtensionLevel 3*) An indirect object reference to a rich media annotation dictionary for an annotation for which to execute the script command.
`TI`	dictionary	(*Optional; ExtensionLevel 3*) A dictionary that shall be an indirect object reference to a RichMediaInstance dictionary that is also present in the `Instances` array of the annotation. **Note:** Because of the potential for multiple `Flash` instances within a single annotation, the `TI` entry allows targeting of actions to a specific `Flash` instance.
`CMD`	dictionary	(*Required; ExtensionLevel 3*) A RichMediaCommand dictionary containing the command name and arguments to be executed when the rich-media-execute action is invoked. See "TABLE 8.48b Entries in a RichMediaCommand dictionary" on page 41.

The *RichMediaCommand dictionary* contains a command name and optional arguments to be passed to the annotation handler specific to the target instance specified by the `TI` key in the parent rich-media-execute action dictionary.

TABLE 8.48b Entries in a RichMediaCommand dictionary

KEY	TYPE	DESCRIPTION
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes; shall be RichMediaCommand for a RichMediaCommand dictionary.
C	text string	(*Required; ExtensionLevel 3*) A text string specifying the script command (a primitive ActionScript or JavaScript function name). If the target instance (specified by the TI key in the parent rich-media-execute action dictionary) is Flash content, the command string represents an ActionScript ExternalInterface call to the script engine context specific to the target instance. If the target instance is a 3D model, the call is made in the global context of the annotation's instance of the 3D JavaScript engine.
A	various	(*Optional; ExtensionLevel 3*) An object that specifies the arguments to the command. The object can either be a single typed value or an array of typed values, each an argument. Valid arguments are objects of type text string, integer, real, or Boolean. Default value: no arguments.

8.6 Interactive Forms

Extension level 1 adds support for the rich text specified in XML Forms Architecture (XFA), versions 2.5 and 2.6. The following entry in Table 8.73 reflects the clarified description from the Errata for the PDF Reference, sixth edition, version 1.7, available at www.adobe.com/go/acrobat_developer. (Select the Documentation *tab.) Unchanged content is shown in gray.*

Table 8.73 Attributes of the <body> element

ATTRIBUTE	DESCRIPTION
`xfa:spec`	The version of the *XML Forms Architecture (XFA)* specification to which the rich text string complies. The following are valid values: • `2.0`, which specifies XFA version 2.0. PDF 1.5 supports this version. • `2.2`, which specifies XFA version 2.2. PDF 1.6 supports this version and the earlier versions back to XFA version 2.0. • `2.4`, which specifies XFA version 2.4. PDF 1.7 supports this version and the earlier versions back to XFA version 2.0. • `2.5`, which specifies XFA version 2.5. This value is an extension added in Acrobat 8.1. (*ExtensionLevel 1*) • `2.6`, which specifies XFA version 2.6. This value is an extension added in Acrobat 8.1. (*ExtensionLevel 2*) See implementation note E-5, page 130.

8.6.3 Field Types

Signature Fields

Add the new `P` *entry to Table 8.82 on page 697 of the PDF Reference.*

TABLE 8.82 Entries in a signature field lock dictionary

KEY	TYPE	DESCRIPTION
`P`	number	(*Optional; ExtensionLevel 3*) The access permissions granted for this document. Valid values follow: `1`, no changes to the document are permitted; any change to the document invalidates the signature. `2`, permitted changes are filling in forms, instantiating page templates, and signing; other changes invalidate the signature. `3`, permitted changes are the same as for `2`, as well as annotation creation, deletion, and modification; other changes invalidate the signature. Default value: none; absence of this key results in no effect on signature validation rules. If MDP permission is already in effect from an earlier incremental save section or the original part of the document, the number shall specify permissions less than or equal to the permissions already in effect based on signatures earlier in the document. That is, permissions can be denied but not added. If the number specifies greater permissions than an MDP value already in effect, the new number is ignored. If the document does not have an author signature, the initial permissions in effect are those based on the number `3`. The new permission applies to any incremental changes to the document following the signature of which this key is part. See implementation note E-14 on page 131.

The following are additions and changes to Table 8.83 on page 697 of the PDF Reference. New entries are `LockDocument` *and* `AppearanceFilter`*. Unchanged content is shown in gray.*

TABLE 8.83 Entries in a signature field seed value dictionary

KEY	TYPE	DESCRIPTION
`LockDocument`	name	(*Optional; ExtensionLevel 3*) A name value supplying the author's intent for whether the signing dialog should allow the user to lock the document at the time of signing. Values are `true`, `false`, and `auto`, as follows: `true`, the document should be locked at the time of signing. If the `Ff` entry indicates that `LockDocument` is not a required contraint, the user may choose to override this at the time of signing; otherwise, the document is locked after signing. `false`, the document should not be locked after signing. Again, the required flag, `Ff`, determines whether this is a required constraint. `auto`, the consuming application decided whether to present the lock user interface for the document and whether to honor the required flag, `Ff`, based on the properties of the document. Default value: `auto` See implementation note E-15 on page 131.
`AppearanceFilter`	text string	(*Optional; ExtensionLevel 3*) A text string naming the appearance to be used when signing the signature field. Conforming readers may choose to maintain a list of named signature appearances. This text string provides authors with a means of specifying which appearance should be used to sign the signature field. If the required bit `AppearanceFilter` in `Ff` is set, the appearance shall be available to sign the document and is used. See implementation note E-16 on page 131.

TABLE 8.83 Entries in a signature field seed value dictionary

KEY	TYPE	DESCRIPTION
Ff	integer	(*Optional*) A set of bit flags specifying the interpretation of specific entries in this dictionary. A value of 1 for the flag indicates that the associated entry is a required constraint. A value of 0 indicates that the associated entry is an optional constraint. Bit positions are 1 (Filter); 2 (SubFilter); 3 (V); 4 (Reasons); 5 (LegalAttestation); 6 (AddRevInfo); 7 (DigestMethod); for extension level 3, the following bit flags are added: 8 (LockDocument); and 9 (AppearanceFilter). Default value: 0
V	real	(Optional) The minimum required capability of the signature field seed value dictionary parser. A value of 1 specifies that the parser must be able to recognize all seed value dictionary entries specified in PDF 1.5. A value of 2 specifies that it must be able to recognize all seed value dictionary entries specified in PDF 1.7 and earlier. A value of 3 specifies that it shall be able to recognize all seed value dictionary entries specified in extension level 3 and earlier. The Ff entry indicates whether this is a required constraint. **Note:** The PDF References fifth edition (PDF1.6) and earlier, erroneously indicates that the V entry is of type integer. This entry is of type real.

Insert the following new section after the subsection titled "Signature Fields" on page 702 of the PDF Reference.

Barcode Fields

A *barcode field* is a text field (field type Tx) in which its appearance is rendered by one or more annotation widgets that contain a PMD entry. (See the *PaperMetaData generation parameters dictionary*, page 46.) The appearance of the barcode field, called the *textual value*, contains the value that is to be recovered from the barcode at scan time. (See implementation note E-6, page 130.)

A barcode field dynamically acquires its value through user input into one or more text fields that are referenced in a calculate dictionary (see the C entry in Table 8.46, "Entries in a form field's additional-actions dictionary" in the *PDF Reference*) of an additional-actions dictionary. (See the AA entry in Table 8.39, "Additional entries specific to a widget annotation" in the *PDF Reference*.)

Additional keys in the form field and widget annotation dictionaries control whether and how the data value is compressed, encrypted, and segmented; which barcode symbology is employed; and any other parameters required for generating the barcode.

Besides the usual entries common to all fields (see Table 8.69 on page 675 of the *PDF Reference*), the field dictionary for a barcode field may contain the additional entry shown in Table 8.39a. (See also implementation note E-7, page 131.)

TABLE 8.39a Additional entry specific to a barcode field

KEY	TYPE	VALUE
DataPrep	number	(*Optional; ExtensionLevel 3*) Describes the data preparation steps before encoding. Permissible values follow: 0 The data is sent directly to the encoder. 1 The data undergoes flate compression before encoding. The default is 0, the data is sent directly to the encoder.

The widget annotation dictionary of a barcode field may be merged with the field dictionary when there is a single barcode associated with the field. If there are several widget annotations associated with a single barcode field, the data is divided into approximately equal parts, and each part is encoded (rendered) separately according to the specifications of the PMD entry in the widget annotation dictionary. (See Table 9.39 on page 39 for the PMD entry.) The PMD entry is a *PaperMetaData generation parameters dictionary*. The entries of a PaperMetaData generation parameters dictionary are listed in Table 8.39b, shown below.

TABLE 8.39b Entries in a PaperMetaData generation parameters dictionary

KEY	TYPE	VALUE
Type	name	(*Required; ExtensionLevel 3*) The type of PDF object that this dictionary describes; shall be PaperMetaData for a *PaperMetaData generation parameters dictionary*.
Version	number	(*Required; ExtensionLevel 3*) Versioning mechanism, for forward compatibility. Should be the number 1.
Resolution	number	(*Optional; ExtensionLevel 3*) The resolution, in dots-per-inch (dpi), at which the barcode object is rendered. Default value is 300.
Caption	text string	(*Optional; ExtensionLevel 3*) The caption of the barcode object. The default value of Caption is the "file: scheme" URL of the current document.
Symbology	name	(*Required; ExtensionLevel 3*) Specifies which barcode or glyph technology is to be used on this annotation. Supported values are PDF417, QRCode, and DataMatrix.
Width	number	(*Required; ExtensionLevel 3*) The width, measured in inches, of the barcode object.
Height	number	(*Required; ExtensionLevel 3*) The height, measured in inches, of the barcode object.

TABLE 8.39b Entries in a PaperMetaData generation parameters dictionary

KEY	TYPE	VALUE
XSymWidth	number	(*Required; ExtensionLevel 3*) The horizontal distance, in pixels, between two barcode modules. Shall be presented as an integer value.
XSymHeight	number	(*Required; ExtensionLevel 3*) Only needed for PDF417. The vertical distance between two barcode modules, measured in pixels. The ratio XSymHeight/XSymWidth shall be an integer value. For PDF417, the acceptable ratio range is from 1 to 4. For QRCode and DataMatrix, this ratio shall always be 1.
ECC	number	(*Required for PDF417 and QRCode only; ExtensionLevel 3*) An integer value representing the error correction coefficient. For PDF417, shall be from 0 to 8. For QRCode, shall be from 0 to 3 (0 for 'L', 1 for 'M', 2 for 'Q', and 3 for 'H').
nCodeWordRow	number	Needed only for PDF417. The number of codewords per barcode row. Defaults to 0. Obsolete.
nCodeWordCol	number	Needed only for PDF417. The number of codewords per barcode column. Defaults to 0. Obsolete.

The Symbology entry takes one of three values, PDF417, QRCode, and DataMatrix. See the listing of these symbology standards in the International Organization for Standardization (ISO) section of the Bibliography.

The XSymWidth entry can be calculated by multiplying the desired distance, in inches, by the Resolution value, and then rounding up the result.

The ECC entry, the *error correction coefficient*, corresponds to the level of data redundancy that is added to the barcode to correct any potential decoding errors. Higher levels provide more redundancy and a more robust barcode that will generate more successful decode results; however, higher levels also result in a larger barcode and a reduced ability to encode user-supplied or form structure data into the barcode.

The DataPrep entry determines whether the value of a computed barcode field is compressed. The encoding (or rendering) of the data occurs whenever the text value of a barcode field changes. Starting with the value of the field as stored in the V key of the barcode field dictionary, the following steps are applied:

1. **Compression.** If DataPrep entry is present and its value is 1, compress the text value of the barcode field using flate compression.

2. **Rendering.** Associate each message with a widget annotation referred to through the Kids array from the barcode field. (If there is only one widget annotation, its widget annotation dictionary is merged into the barcode field dictionary.) For each such association, the PMD entry (in the PaperMetaData generation parameters dictionary) of the widget annotation instructs which barcode or glyph symbology to use, and the parameters to specify how to generate the barcode or glyph. An appearance stream for the widget (the N entry in the AP dictionary) is then constructed that causes the entire rectangular area of the widget to be filled with the generated image.

8.6.7 XFA Forms

Place the following material at the end of this section.

Incorporation of XFA Datasets into a PDF/A-2 Conforming File

To support PDF/A-2 conforming files, ExtensionLevel 3 adds support for XML form data (XFA datasets) through the `XFAResources` name tree, which is part of the name dictionary of the document catalog. (See "TABLE 3.28 Entries in the name dictionary" on page 23.) While Acrobat forms (and form data) are permitted in a PDF/A-2 conforming file, XML forms are not. Such XML forms are specified as XDP streams referenced from interactive form dictionaries. XDP streams can contain XFA datasets.

For applications that convert PDF documents to PDF/A-2, the `XFAResources` name tree supports relocation of XML form data from XDP streams in a PDF document into the `XFAResources` name tree.

The `XFAResources` name tree consists of a string name and an indirect reference to a stream. The string name is created at the time the document is converted to a PDF/A-2 conforming file. The stream contains the `<xfa:datasets>` element of the XFA, comprised of `<xfa:data>` elements.

In addition to data values for XML form fields, the `<xfa:data>` elements enable the storage and retrieval of other types of information that may be useful for other workflows, including data that is not bound to form fields, and one or more XML signature(s).

See the *XML Architecture, XML Forms Architecture (XFA) Specification, version 2.6* in the Bibliography.

8.8 Measurement Properties

Add new entries `Measure` and `PtData` to Table 8.109 in support of geospatial content. Unchanged content is shown in gray. See Section 8.8.1 Geospatial Features, on page 49.

TABLE 8.109 Entries in a viewport dictionary

KEY	TYPE	VALUE
`Measure`	dictionary	(*Optional*) A measure dictionary (see Table 8.110, page 49) that specifies the scale and units that apply to the image.
`PtData`	dictionary	(*Optional; ExtensionLevel 3*) A point data dictionary (see Table 111d, page 53) that specifies the extended geospatial data that applies to the image.

On page 745, modify the following paragraph to reflect the new geospatial coordinate system. Unchanged content appears in gray.

Table 8.110 shows the entries in a measure dictionary. PDF 1.6 defines only a single type of coordinate system, a *rectilinear* coordinate system, specified by the value `RL` for the `Subtype` entry, which is defined as one in which the x and y axes are perpendicular and have units that increment linearly (to the right and up, respectively). Extension level 3 defines a *geospatial* coordinate system specified by the value `GEO` for the `Subtype` entry. Other subtypes are permitted, providing the flexibility to measure using other types of coordinate systems.

When the value of the Subtype entry is GEO, the dictionary defines the relationship between points or regions in the two dimensional PDF object space and points or regions with respect to an underlying model of the earth (or, potentially, other ellipsoid objects).

Add the value of GEO to the Subtype entry of Table 8.110.

TABLE 8.110 Entries in a measure dictionary

KEY	TYPE	VALUE
Subtype	name	(*Optional*) A name specifying the type of coordinate system to use for measuring. Valid values follow: RL, for a rectilinear coordinate system GEO, (*ExtensionLevel 3*) for a geospatial coordinate system Default value: RL

Between the two paragraphs that follow Table 8.110, insert the following.

When the subtype of a measurement dictionary is GEO, additional entries are defined. Table 8.111a, page 50, lists and describes these additional entries in a *geospatial measure dictionary*.

Add a new section to the end of Section 8.8.

8.8.1 Geospatial Features

PDF is a common delivery mechanism for map and satellite imagery data. In extension level 3, a geospatial coordinate system is introduced (Table 8.110, page 49) along with a number of PDF constructs, as explained in this section, to support geospatially registered content.

Geospatial Measure Dictionary

When the subtype of a measurement dictionary (Table 8.110, page 49) is GEO, additional entries are defined through a *geospatial measure dictionary*.

A *geospatial measure dictionary*, Table 111a, contains a description of the earth-based coordinate system associated with the PDF object, and corresponding arrays of points in that coordinate system and the local object coordinate system. It may contain a bounding polygon (the Bounds entry), which defines the region of the PDF object for which the geographic associations are valid. It may also contain a choice of default units (the PDU entry) for user displays of positions, distances and areas. An optional display coordinate system (the DCS entry) allows a document to be authored to display values in a coordinate system other than that associated with the source data. For example, a map may be created in a state plane coordinate system based on a 1927 datum, but it is possible to display its latitude and longitude values in the WGS84 datum corresponding to values reported by a GPS device.

The entries of a geospatial measure dictionary are shown in Table 8.111a.

TABLE 8.111a Additional entries specific in a geospatial measure dictionary

KEY	TYPE	VALUE
Bounds	array	(*Optional; ExtensionLevel 3*) An array of numbers taken pairwise that define a series of points that describes the bounds of an area for which geospatial transformations are valid. For maps, this bounding polygon is know as a *neatline*. These numbers are expressed relative to a unit square that describes the BBox associated with a Viewport or form XObject, or the bounds of an image Xobject. If not present, the default values define a rectangle describing the full unit square, with values of [0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0]. **Note:** The polygon description need not be explicitly closed by repeating the first point values as a final point.
GCS	dictionary	(*Required; ExtensionLevel 3*) A projected or geographic coordinate system dictionary.
DCS	dictionary	(*Optional; ExtensionLevel 3)* A projected or geographic coordinate system to be used for the display of position values, such as latitude and longitude. Formatting the displayed representation of these values is controlled by the conforming reader.
PDU	array	(*Optional; ExtensionLevel 3*) Preferred Display Units. An array of three names that identify in order a linear display unit, an area display unit, and an angular display unit. The following are valid linear display units: M, a meter KM, a kilometer FT, an international foot USFT, a U.S. Survey foot MI, an international mile NM, an international nautical mile The following are valid area display units: SQM, a square meter HA, a hectare (10,000 square meters) SQKM, a square kilometer SQFT, a square foot A, an acre SQMI, a square mile The following are valid angular display units: DEG, a degree GRD, a grad (1/400 of a circle, or 0.9 degrees)

TABLE 8.111a Additional entries specific in a geospatial measure dictionary

KEY	TYPE	VALUE
GPTS	array	(*Required; ExtensionLevel 3*) An array of numbers taken pairwise, defining points in geographic space as degrees of latitude and longitude. These values are based on the geographic coordinate system described in the GCS dictionary. (Note that any projected coordinate system includes an underlying geographic coordinate system.)
LPTS	array	(*Optional; ExtensionLevel 3*) An array of numbers taken pairwise that define points in a 2D unit square. The unit square is mapped to the rectangular bounds of the viewport, image XObject, or forms XObject that contain the measure dictionary. This array contains the same number of number pairs as the GPTS array; each number pair is the unit square object position corresponding to the geospatial position in the GPTS array.

Geographic Coordinate System Dictionary

A geographic coordinate system (GEOGCS) specifies an ellipsoidal object in geographic coordinates: angular units of latitude and longitude. The geographic coordinate system can be described in either or both of two well-established standards: as a numeric EPSG reference code, or as a Well KnownText (WKT) string, which contains a description of algorithms and parameters needed for transformations.

Table 8.111b lists the entries in a *geographic coordinate system dictionary*. A geographic coordinate system dictionary may be a value of the GCS or the DCS entry of a *geospatial measure dictionary*. (See Table 8.111a on page 50.*)*

TABLE 8.111b Entries in a geographic coordinate system dictionary

KEY	TYPE	VALUE
Type	name	(*Required; ExtensionLevel 3*) The type of PDF object that this dictionary describes; shall be GEOGCS for a geographic coordinate system dictionary.
EPSG	integer	(*Optional; ExtensionLevel 3*) An EPSG reference code specifying the geographic coordinate system.
WKT	ASCII string	(*Optional; ExtensionLevel 3*) A string of Well Known Text describing the geographic coordinate system. See implementation note E-8, page 131.

Either an EPSG code or a WKT string is required in a geographic coordinate system dictionary. (See implementation note E-20, page 132.)

The EPSG reference codes are described in a database available through www.epsg.org, as administered by the International Association of Oil and Gas Producers (OGP). Well Known Text is specified in document 01-009, *OpenGIS Implementation Specification: Coordinate Transformation Services*, of the Open Geospatial Consortium. (See Bibliography.) See implementation note E-9, page 131.

Examples of WKT description strings appear at the end of the next section, beginning on page 52.

Projected Coordinate System Dictionary

A projected coordinate system (PROJCS), which includes an embedded GEOGCS, specifies the algorithms and associated parameters used to transform points between geographic coordinates and a two-dimensional (projected) coordinate system. Any transformation between a three-dimensional curved geographic coordinate system and a two-dimensional coordinate system introduces distortions. For small areas, this distortion may be small enough to allow direct mapping between geographic coordinates and PDF object coordinates without requiring the use of a projected coordinate system.

The projected coordinate system can be described in either or both of two well-established standards: as a numeric EPSG reference code, or as a Well KnownText (WKT) string, which contains a description of algorithms and parameters needed for transformations.

The EPSG reference codes are described in a database available through www.epsg.org, as administered by the International Association of Oil and Gas Producers (OGP). Well Known Text is specified in document 01-009, *OpenGIS Implementation Specification: Coordinate Transformation Services*, of the Open Geospatial Consortium. (See Bibliography.) See implementation note E-9, page 131.

Table 8.111c lists the entries in a *projected coordinate system dictionary*. A projected coordinate system dictionary may be a value of the `GCS` or the `DCS` entry of a *geospatial measure dictionary,* Table 8.111a on page 50.

TABLE 8.111c Entries in a projected coordinate system dictionary

KEY	TYPE	VALUE
`Type`	name	(*Required; ExtensionLevel 3*) The type of PDF object that this dictionary describes; shall be `PROJCS` for a projected coordinate system dictionary.
`EPSG`	integer	(*Optional; ExtensionLevel 3*) An EPSG reference code specifying the projected coordinate system.
`WKT`	ASCII string	(*Optional; ExtensionLevel 3*) A string of Well Known Text describing the projected coordinate system. See implementation note E-8, page 131.

Either an `EPSG` code or a `WKT` string is required in the projected coordinate system dictionary. (See implementation note E-20, page 132.)

Example: *A WKT describing a geographic coordinate system*

An example of WKT description of a geographic coordinate system, formatted for readability. The EPSG code equivalent to the GCS_North_American_1983 geographic coordinate system is 4269.

```
GEOGCS["GCS_North_American_1983",
   DATUM["D_North_American_1983",
      SPHEROID["GRS_1980",6378137.0,298.257222101]
   ],
   PRIMEM["Greenwich",0.0],
   UNIT["Degree",0.0174532925199433]
]
```

Example: *A WKT describing a projected coordinate system*

An example of WKT description of a projected coordinate system, formatted for readability. The EPSG code equivalent to the North_American_Albers_Equal_Area_Conic projected coordinate system is 102008.

```
PROJCS["North_America_Albers_Equal_Area_Conic",
    GEOGCS["GCS_North_American_1983",
        DATUM["D_North_American_1983",
            SPHEROID["GRS_1980",6378137.0, 298.257222101]
        ],
        PRIMEM["Greenwich",0.0],
        UNIT["Degree",0.0174532925199433]
    ],
    PROJECTION["Albers"],
    PARAMETER["False_Easting",0.0],
    PARAMETER["False_Northing",0.0],
    PARAMETER["Central_Meridian",-96.0],
    PARAMETER["Standard_Parallel_1",20.0],
    PARAMETER["Standard_Parallel_2",60.0],
    PARAMETER["Latitude_Of_Origin",40.0],
    UNIT["Meter",1.0]
]
```

Point Data Dictionary

Any 2D object (Viewport, Image XObject, or Form XObject) that contains a measure dictionary (Table 8.110, page 49) of subtype `GEO` can optionally include a `PtData` entry. The value of a `PtData` entry is a *point data dictionary* or an array of point data dictionaries of extended data associated with points in the 2D space. Table 8.111d lists the entries of a point data dictionary.

TABLE 8.111d Entries in a point data dictionary

KEY	TYPE	VALUE
`Type`	name	(*Required; ExtensionLevel 3*) The type of PDF object that this dictionary describes; shall be `PtData` for a point data dictionary.
`Subtype`	name	(*Required; ExtensionLevel 3*) Shall be `Cloud`.

TABLE 8.111d Entries in a point data dictionary

KEY	TYPE	VALUE
Names	array	(*Required; ExtensionLevel 3*) An array of names that identify the internal data elements of the individual point arrays in the XPTS array. There are three predefined names: LAT, latitude in degrees. The XPTS value is a number type. LON, longitude in degrees. The XPTS value is a number type. ALT, altitude in meters. The XPTS value is a number type. **Note:** These names are, in effect, column headers for the array of XPTS values.
XPTS	array	(*Required; ExtensionLevel 3*) An array of arrays of values. The number of members in each interior array corresponds to the size of the Names array; each member in the interior arrays is of a type defined by the corresponding name in the Names array. The XPTS array is a collection of tuples without any guaranteed ordering or relationship from point to point.

The names LAT, LON, and ALT are predefined, and are used to associate altitude information with latitude and longitude positions.

Multimedia Features (Chapter 9 in PDF Reference)

This section describes extensions to the PDF specification contained in the *PDF Reference, sixth edition, version 1.7* and in ISO 32000 (*Document management - Portable document format - PDF 1.7*). The latter document is the version of the PDF specification that has been ratified by the ISO. These documents differ as described in *About the ISO Draft of the PDF 1.7 Reference*.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

9.5 3D Artwork

On page 790, append the indicated sentence to the bullet that begins, "3D stream contain …". Unchanged content is shown in gray.

The following sections describe the major PDF objects that relate to 3D artwork, as well as providing background information on 3D graphics:

- 3D annotations provide a virtual camera through which the artwork is viewed. (see section 9.5.1 [ISO section 13.6.2], "3D Annotations").
- 3D streams contain the actual specification of a piece of 3D artwork (see section 9.5.2 ISO section 13.6.3], "3D Streams"). This specification supports the standard ECMA-363, Universal 3D file format developed by the 3D Industry Forum (see Bibliography). Extension level 1 extends PDF to support the PRC file format (see Bibliography).

9.5.1 3D Annotations

Add the `3DU` entry to Table 9.33.

TABLE 9.33 Additional entries specific to a 3D annotation

KEY	TYPE	VALUE
`3DU`	dictionary	(*Optional; ExtensionLevel 3*) A *3D units dictionary* that specifies the units definitions for the 3D data associated with this annotation. See "TABLE 9.33a Entries in a 3D units dictionary" on page 60.

Add the `Style`, `Window`, and `Transparent` entries to Table 9.34.

TABLE 9.34 Entries in a 3D activation dictionary

KEY	TYPE	VALUE
Style	name	(*Optional; ExtensionLevel 3*) Values may be Embedded or Windowed. A conforming interactive reader supports both Embedded and Windowed. Default value: Embedded
Window	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaWindow Dictionary that describes the size and position of the floating user interface window when the value for Style is set to Windowed. See "RichMediaWindow Dictionary" on page 84 for a detailed description.
Transparent	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates whether the page content is displayed through the transparent areas of the rich media content (where the alpha value is less than 1.0). Default value: false

Add the following commentary to the end of Section 9.5.1.

The Style and Window entries provide the mechanism to present the 3D content in a floating window.

The Transparent entry provides a flag for specifying whether the background of a 3D view is rendered using transparency. If this is the case, some area of the page may be visible through the background if the resulting alpha value at that location is less than 1.0. See also the Transparent entry of the "RichMediaPresentation Dictionary" on page 82.

9.5.2 3D Streams

Modify the Subtype entry as shown in Table 9.35.

TABLE 9.35 Entries in a 3D stream dictionary

KEY	TYPE	VALUE
Subtype	name	(*Required*) A name specifying the format of the 3D data contained in the stream. The following valid values are supported: U3D, which specifies the Universal 3D file format. PRC, which specifies the PRC file format. This value is an extension supported by Acrobat 8.1. (*ExtensionLevel 1)*

On page 798, delete the sentence that begins "The only valid value …" in the following paragraph. The material in that sentence is covered in Table 9.35298], as amended by this guide.

The Subtype entry specifies the format of the 3D stream data. ~~The only valid value is U3D, which indicates that the stream data conforms to the Universal 3D File Format specification~~ (see Bibliography). PDF consumer applications must be prepared to encounter unknown values for Subtype and recover appropriately, which usually means leaving the annotation in its inactive state, displaying its normal appearance.

9.5.3 3D Views

Add the MA entry in Table 9.39 of the PDF Reference.

TABLE 9.39 Entries in a 3D view dictionary

KEY	TYPE	VALUE
MA	array	(*Optional; ExtensionLevel 3*) An array of 3D measurement/markup dictionaries, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. See "3D Measurement/Markup Dictionary" on page 62 for the definition of a 3D measurement dictionary.

Modify the MS entry as shown, and add the MA entry in Table 9.39 of the PDF Reference.

TABLE 9.39 Entries in a 3D view dictionary

KEY	TYPE	VALUE
MS	name	(*Optional*) A name specifying the entry to use for the 3D camera-to-world transformation matrix. The following values are supported: M indicates that the C2W entry specifies the matrix U3D indicates that the U3DPath entry in the 3D stream object is used for the matrix. This value reflects the sole supported value of the Subtype entry in the 3D stream dictionary. **Note:** There is no corresponding MS field value for the PRC file format, that would correspond to a 3D stream object of type PRC. M is the only valid entry for 3D stream objects of type PRC (or it may be omitted). If omitted, the view specified in the 3D artwork is used.
MA	array	(*Optional; ExtensionLevel 3*) An array of 3D measurement/markup dictionaries, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. See "3D Measurement/Markup Dictionary" on page 62 for the definition of a 3D measurement dictionary.

9.5.3 3D Views (Node Dictionaries)

Modify the N entry and add three new entries (Instance, Data, and RM) in Table 9.47 as follows.

TABLE 9.47 Entries in a 3D node dictionary

KEY	TYPE	VALUE
N	text string	(*Required*) The name of the node being described by the node dictionary. Interpretation of this entry depends upon the 3D format specified in the Subtype entry in Table 9.35, as described below: • U3D. If the Subtype of the corresponding 3D Stream is U3D, this entry corresponds to the field Node block name, specified in the *Universal 3D File Format* (See Bibliography.) • PRC. (*ExtensionLevel 1*) If the Subtype of the corresponding 3D Stream is PRC, this entry is constructed from fields stored in the PRC stream. For information on deriving names from PRC fields, see the documentation module "Naming PRC entities for outside PDF referencing" in the *PRC Format* (see Bibliography). **Note:** When comparing this entry to node names for a particular convention, such as the Universal 3D File Format, conforming readers translate between the PDF text encoding used by PDF and the character encoding specified in the 3D stream.
Instance	dictionary	(*Required if Data is present; ExtensionLevel 3*) An indirect object reference to a RichMediaInstance dictionary (see "RichMediaInstance Dictionary" on page 88) that is also referenced by an entry in the Instances array.
Data	text string or stream	(*Optional; ExtensionLevel 3*) A text string or stream that contains state data to be passed to the instance when the view is triggered. See the extended description of the Data key in "View Params Dictionary" on page 93.
RM	dictionary	(*Optional; ExtensionLevel 3*) A render mode dictionary that specifies the render mode and related properties for this node. If omitted, the render mode specified in the 3D view is used, and if that is not present, the render mode of the 3D artwork is used. The render mode dictionary is identical to that used by the 3D view dictionary. See the Section 9.5.3, "3D Views (3D Render Mode Dictionaries)" in *PDF Reference*.

Add the following commentary following the paragraph that begins with "The M entry specifies…" on page 830 of the PDF Reference.

The Instance and Data entries provide a reference to an instance array dictionary within the Instances array. The content of Data is passed to and from the Flash run time in a rich media context.

The RM provides the facility of specifying a change in render mode for each node. In PDF 1.7, render mode could be specified globally or per view. See Section 9.5.3 "3D Views" of the *PDF Reference*.

The RM entry to the 3D node dictionary is applicable both for the 3D node structure within a rich media context (see Section 9.6.1 RichMedia Annotations on page 76) and within the existing 3D annotations structure. (See Section 9.5.1 "3D Annotations" of the *PDF Reference* for further information.] The Instances array and additional Data entries are applicable only to 3D nodes within a rich media context

Add the next section after Section 9.5.5.

9.5.6 Persistence of 3D Measurements and Markups

Beginning with extension level 3, users can add 3D measurement data to an instance of a 3D artwork. This measurement data is stored in a 3D measurement/markup dictionary. 3D measurements are associated with 3D views, and each 3D view can contain zero or more 3D measurement dictionaries.

After a measurement is associated with view, it is visible only when that view is selected or active. As different views are displayed, the measurements associated with that view are made visible and previously displayed measurements become invisible.

3D measurements can either be simple 3D markups used to add information to the geometric data shown in a view, or have an associated comment, in which case they have all the associated functionality of a comment. A 3D measurement can be promoted or demoted to or from comment status.

There are three key aspects to defining persistent 3D measurements:

- A mechanism to define the units associated with the geometric data being measured. A `3DU` entry in TABLE 9.33 Additional entries specific to a 3D annotation, page 55, has as its value a 3D units dictionary, which stores the units data for this 3D annotation. [See "The 3D Units Dictionary" on page 59 for more information.]
- The association between 3D measurements and 3D view is realized through the `MA` entry in a 3D view dictionary ("TABLE 9.39 Entries in a 3D view dictionary" on page 57). The value of this entry is an array of *3D measurement/markup dictionaries*, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. For more information about 3D measurement dictionaries, see "3D Measurement/Markup Dictionary" on page 62.
- When a 3D measurement is promoted to a comment, a *projection annotation* ("Projection Annotations" on page 39) is created to manage the comment and its appearance in the comments list. An indirect reference to this projection annotation is placed in the 3D measurement dictionary. [See "3D Measurements and Projection Annotations" on page 75.]

The 3D Units Dictionary

The data associated with a 3D artwork annotation may be defined in an arbitrary 3D coordinate system. For viewing purposes, the application defines a camera to map these coordinates onto a view surface. (3D coordinate systems are discussed in Section 9.5.4 of the *PDF Reference*.) For measurement purposes, distances are computed in this arbitrary coordinate system and assigned physical meaning by entries in the *3D units dictionary*. These sets of optional units should be defined:

- **Creation time units:** Units known at the time the 3D artwork annotation is created.
- **User override units:** Units defined by the user after the annotation was created
- **Display units:** Units that the user would like used when displaying distances for all newly created measurements.

The first two definitions assign physical meaning to measured distances, and the third defines how the distances are presented. When a 3D annotation is created, the application may have information from external sources that allows it to determine the units of the data being imported. Later the user may want to either override that definition or control what units data is displayed in.

In addition to defining the units, a scaling operation is defined that maps one set of units to another. For the creation time and user override units, the mapping states that "m model data units = n real units". For

the display units, the mapping states that "m model units = n display units". In most cases, the m and n scale values will be 1.0 because most data is defined in some well-known units system, such as meters or inches.

The entries in the 3D units dictionary establish these mappings for each set of units (creation time, user override, and display units). The 3D Units dictionary is referenced in TABLE 9.33 Additional entries specific to a 3D annotation as the value of the 3DU entry.

TABLE 9.33a Entries in a 3D units dictionary

KEY	TYPE	VALUE
TSm	number	(*Optional; ExtensionLevel 3*) The creation time units m scale value. If omitted, TSm defaults to 1.0; if included, TU shall exist.
TSn	number	(*Optional; ExtensionLevel 3*) The creation time units n scale value. If omitted, TSn defaults to 1.0; if included, TU shall exist.
TU	text string	(*Optional; ExtensionLevel 3*) The creation time units value. A text string specifying a label for displaying the units represented by this dictionary in a user interface. It is recommended that the label use a universally recognized abbreviation.
USm	number	(*Optional; ExtensionLevel 3*) The user defined units m scale value. If omitted, USm defaults to 1.0; if included, UU shall exist.
USn	number	(*Optional; ExtensionLevel 3*) The user defined units n scale value. If omitted, USn defaults to 1.0; if included, UU shall exist.
UU	text string	(*Optional; ExtensionLevel 3*) The user override units value. A text string specifying a label for displaying the units represented by this dictionary in a user interface. It is recommended that the label use a universally recognized abbreviation.
DSm	number	(*Optional; ExtensionLevel 3*) The display units m scale value. If omitted, DSm defaults to 1.0; if included, DU shall exist.
DSn	number	(*Optional; ExtensionLevel 3*) The display units n scale value. If omitted, DSn defaults to 1.0; if included, DU shall exist.
DU	text string	(*Optional; ExtensionLevel 3*) The display units value. A text string specifying a label for displaying the units represented by this dictionary in a user interface. It is recommended that the label use a universally recognized abbreviation.

The following algorithm is used to map model space distances for display in 3D measurements for each of the three sets of units.

The following are default values:

```
n = 1.0                    // a number
m = 1.0                    // a number
Units = "Model Units"      // a text string
```

In the algorithm that follows, an entry is *defined* if it is included in the 3D Units dictionary.

Creation time units – The following is the process creation time units definition:

If `TU` is defined, then `Units = TU`
If `TSm` is defined, then `m = TSm`
If `TSn` is defined, then `n = TSn`

Note: If either `TSm` or `TSn` is defined, `TU` should be included in the 3D Units dictionary; if `TU` is not defined in this case, the unit specification is undefined and is ignored.

User override units – The following is the process user override units definition:

If `UU` is defined, then `Units = UU`
If `USm` is defined, then `m = USm`
If `USn` is defined, then `n = USn`

Note: If either `USm` or `USn` is defined, `UU` should be included in the 3D Units dictionary; if `UU` is not defined in this case, the unit specification is undefined and is ignored.

Display units – The following is the display units definition:

If `DU` is defined, then `Units = DU`
If `DSm` is defined, then `m = m * DSm`
If `DSn` is defined, then `n = n * DSn`

Note: If either `DSm` or `DSn` is defined, `DU` should be included in the 3D Units dictionary; if `DU` is not defined in this case, the unit specification is undefined and is ignored.

Finally, if X is a model space distance and Y is the displayed value, the functional relationship between X and Y is given by the equation `Y Units = (m/n) * X`.

3D Measurement/Markup Dictionary

The MA entry of TABLE 9.39 Entries in a 3D view dictionary contains an array of 3D measurement dictionaries, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. Each 3D measurement dictionary describes the type of the 3D measurement through the Subtype entry, as well as provides a name for the measurement as it may appear in the user interface of a conforming reader.

The entries of the 3D Measurement/Markup dictionary are described in the table that follows.

TABLE 9.39a Entries in a 3D measurement/markup dictionary common to all markup subtypes

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes; if present, it shall be 3DMeasure for a 3D measurement dictionary.
Subtype	name	(*Required; ExtensionLevel 3*) A name specifying the measurement type for this measurement: • LD3 – A *linear dimension measurement* is used to denote the distance between two arbitrary points on a 3D model. See "3D Linear Dimension Measurement" on page 63 for a listing of additional entries in this dictionary. • PD3 – A *perpendicular dimension measurement* is used to denote the perpendicular distance between two geometric entities (normally two lines or a point and a line). See "3D Perpendicular Dimension Measurement" on page 65 for a listing of additional entries in this dictionary. • AD3 – An *angular dimension measurement* is used to denote the angle between two linear entities. See "3D Angular Dimension Measurement" on page 67 for a listing of additional entries in this dictionary. • RD3 – A radial dimension measurement is used to define the radius or diameter of a circular 3D entity. See "3D Radial Dimension" on page 71 for a listing of additional entries in this dictionary. • 3DC – A 3D comment note lets users connect a comment to a specific piece of geometry in the 3D model. See "3D Comment Note" on page 74 for a listing of additional entries in this dictionary.
TRL	text string	(*Optional; ExtensionLevel 3*) A name string that may be associated with a measurement markup. If omitted, a conforming interactive viewer may create one. Conforming viewers that do not provide a user interface for such elements should ignore this field.

The TRL field contains the current name for the measurement markup. Each measurement markup is assigned a name ("Measurement 1", "3D Comment 22", …and so on) when it is created. These names may be shown by an interactive conforming reader so that users can see what measurements are associated with what views. Users can override the initially defined names at any point.

3D Linear Dimension Measurement

A *3D linear measurement* is a markup showing the distance between two arbitrary points on a 3D model. Here is an example.

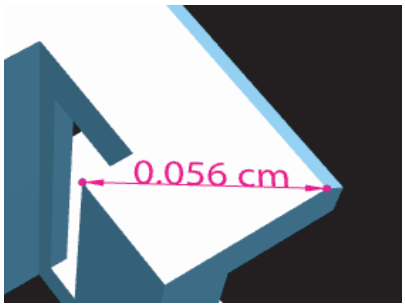

FIGURE 9.18

As shown, a *3D linear measurement* consists of two filled circles, called *anchor points*, one at each of the two positions being measured, and a line with an arrowhead on each end connecting the two anchor points (referred to as the measure line). This line is then labeled with a value representing the distance between the two anchor points.

In addition to the entries in "TABLE 9.39a Entries in a 3D measurement/markup dictionary" on page 62, the following entries are defined for a 3D measurement dictionary with a `Subtype` value of `LD3` for 3D linear measurement.

TABLE 9.39b Additional entries in a 3D measurement/markup dictionary for a 3D linear dimension measurement

KEY	TYPE	VALUE
`AP`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup will lie.
`A1`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the first anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
`N1`	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 1 (`A1`). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
`A2`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the second anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
`N2`	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 2 (`A2`). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
`TP`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the text anchor point for the measurement value string.

TABLE 9.39b Additional entries in a 3D measurement/markup dictionary for a 3D linear dimension measurement

KEY	TYPE	VALUE
TY	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the up direction vector, called the text Y direction, for the text string presenting the measurement value string.
TS	number	(*Optional; ExtensionLevel 3*) A number representing the measurement text string height defined in points in the default user space. Note that measurement text is zoom invariant. The default is 12 points.
C	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0 that represent the RGB color of the measurement markup. The default value is the array [1 1 1], representing the color white.
V	number	(*Required; ExtensionLevel 3*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
U	text string	(*Required; ExtensionLevel 3*) A string, called the units string, that represents the units for the measurement.
P	integer	(*Optional; ExtensionLevel 3*) The number of decimal digits, which represents the precision, shown for the measurement value (V). The default is 3, if P is not specified.
UT	text string	(*Optional; ExtensionLevel 3*) A string defined by the user that is appended to the end of the measurement value string. If omitted, no string is appended.
S	dictionary	(*Optional; ExtensionLevel 3*) A comment reference is an *indirect reference* to a projection annotation that may be associated with this 3D measurement. See "3D Measurements and Projection Annotations" on page 75.

FIGURE 9.19 depicts some of the parameters for a 3D linear measurement.

The value text is drawn on the annotation plane (AP) where the horizontal text path is defined by the vector from A1 to A2 with the text up direction defined by the vector TY. The lower left corner of the text box is positioned at the text anchor point (TP).

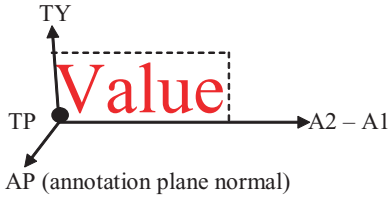

FIGURE 9.19

The actual Y-axis is formed by taking the cross product of AP and (A2 - A1) the vector TY is used only to determine the orientation of this Y-axis.

If the text position TP is outside the area between A1 and A2, an extension line collinear to the measure line connecting TP to the closest anchor point is generated.

There are three parts to the text string displayed with the measurement, a numeric value (V), a units string (U), and an optional user string (UT). The display of the numeric value field number is also controlled by the precision value (P), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the units string and user text as appropriate. This process is viewer dependent.

3D Perpendicular Dimension Measurement

A *perpendicular measurement* is used to denote the perpendicular distance between two geometric entities (normally two lines or a point and a line) as illustrated here.

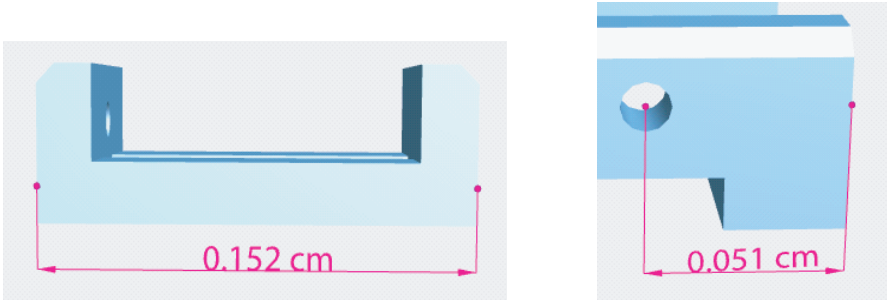

FIGURE 9.20

In these figures, a perpendicular measurement markup consists of two filled circles at the anchor points, two parallel extension lines (referred to as leader lines) starting at the anchor points and extending away from the anchor points. There is also a labeled line with arrowheads on both sides (referred to as the measure line) indicating that the distance shown is the perpendicular distance between the two parallel lines.

In addition to the entries in "TABLE 9.39a Entries in a 3D measurement/markup dictionary" on page 62, the following entries are defined for a 3D measurement dictionary with a Subtype value of PD3 for 3D perpendicular measurement.

TABLE 9.39c Additional entries in a 3D measurement/markup dictionary for a 3D perpendicular dimension measurement

KEY	TYPE	VALUE
AP	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup will lie.
A1	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the first anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
N1	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 1 (A1). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.

TABLE 9.39c Additional entries in a 3D measurement/markup dictionary for a 3D perpendicular dimension measurement

KEY	TYPE	VALUE
A2	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the second anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
N2	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 2 (A2). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
D1	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the direction vector for leader lines associated with the anchor points (A1 and A2).
TP	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TY	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the up direction vector, called the text Y direction, for the text string presenting the measurement value string.
TS	number	(*Optional; ExtensionLevel 3*) A number representing the measurement text string height defined in points in the default user space. Note that measurement text is zoom invariant. The default is 12 points.
C	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing the RGB color of the measurement markup. The default value is the array [1 1 1], representing the color white.
V	number	(*Required; ExtensionLevel 3*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
U	text string	(*Required; ExtensionLevel 3*) A string, called the units string, representing the units for the measurement.
P	integer	(*Optional; ExtensionLevel 3*) The number of decimal digits, which represents the precision, shown for the measurement value (V). The default is 3, if P is not specified.

TABLE 9.39c Additional entries in a 3D measurement/markup dictionary for a 3D perpendicular dimension measurement

KEY	TYPE	VALUE
UT	text string	(*Optional; ExtensionLevel 3*) A string defined by the user that is appended to the end of the measurement value string. If omitted, no string is appended.
S	dictionary	(*Optional; ExtensionLevel 3*) A comment reference is an *indirect reference* to a projection annotation that may be associated with this 3D measurement. See "3D Measurements and Projection Annotations" on page 75.

The following figure illustrates the parameters associated with the perpendicular dimension.

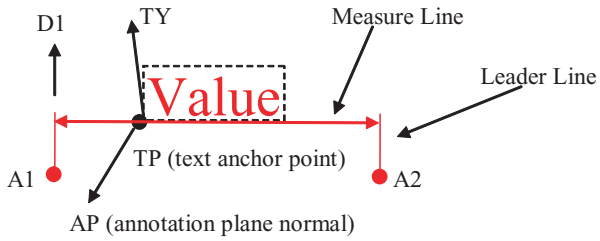

FIGURE 9.21

FIGURE 9.21 shows the measure markup and parameters. All the markup items are drawn on the annotation plane (as defined by AP). The text layout is defined in a similar manner as for linear dimensions. The lower-left corner of the text box is positioned at the text anchor point (TP), and the text's X-axis is aligned with the measure line. The text will flow in the direction defined by a vector from A1 to A2. The text's up direction is defined as the cross product of the annotation plane normal and the text X-axis, in the direction defined by the TY parameter.

In addition to controlling text position, the text anchor point (TP) also controls the lengths of the leader lines and the placement of the measure line. Because the leader lines are parallel and the measure line must be perpendicular to both leader lines, the intersection of the leader lines and the measure line is easily computed.

If the text position TP is outside the area between A1 and A2, an extension line collinear to the measure line connecting TP to the closest anchor point is generated.

There are three parts to the text string displayed with the measurement: a numeric value (V), a units string (U), and an optional user string (UT). The display of the numeric value field number is also controlled by the precision value (P), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the units string and user text as appropriate. This process is viewer dependent.

3D Angular Dimension Measurement

An *angular measurement* is used to denote the angle between two linear entities, as shown here.

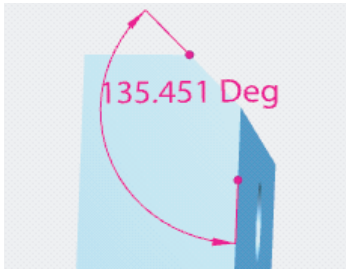

FIGURE 9.22

An angular measurement markup consists of two anchor points, FIGURE 9.22, one located on each of the two linear entities whose angle is being measured. Connected to each anchor point is an extension line that is collinear with the edge it measures. A labeled arc, with an arrowhead at each end, connects the two extension lines, making it clear which angle is being measured.

In addition to the entries in "TABLE 9.39a Entries in a 3D measurement/markup dictionary" on page 62, the following entries are defined for a 3D measurement dictionary with a `Subtype` value of `AD3` for 3D angular measurement.

TABLE 9.39d Additional entries in a 3D measurement/markup dictionary for a 3D angular dimension measurement

KEY	TYPE	VALUE
`AP`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup lies.
`A1`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the first anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
`D1`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the direction vector for the leader line associated with the first anchor point (`A1`).
`N1`	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 1 (`A1`). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
`A2`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the second anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
`D2`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the direction vector for the leader line associated with the second anchor point (`A2`).

TABLE 9.39d Additional entries in a 3D measurement/markup dictionary for a 3D angular dimension measurement

KEY	TYPE	VALUE
N2	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 2 (A2). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
TP	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TX	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the horizontal direction vector for the text string presenting the measurement value string.
TY	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the up direction vector, called the text Y direction, for the text string presenting the measurement value string.
TS	number	(*Optional; ExtensionLevel 3*) A number representing the measurement text string height defined in points in the default user space. Note that measurement text is zoom invariant. The default is 12 points.
C	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing the RGB color of the measurement markup. The default value is the array [1 1 1], representing the color white.
V	number	(*Required; ExtensionLevel 3*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
P	integer	(*Optional; ExtensionLevel 3*) The number of decimal digits, which represents the precision, shown for the measurement value (V). The default is 3, if P is not specified.
UT	text string	(*Optional; ExtensionLevel 3*) A string defined by the user that is appended to the end of measurement value string. If omitted, no string is appended.
DR	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates whether degrees or radians are shown in angular measurements. If DR is true, angular measurements are shown in degrees. The default is true.
S	dictionary	(*Optional; ExtensionLevel 3*) A comment reference is an *indirect reference* to a projection annotation that may be associated with this 3D measurement. See "3D Measurements and Projection Annotations" on page 75.

The key geometric parameters for an angular dimension are shown here.

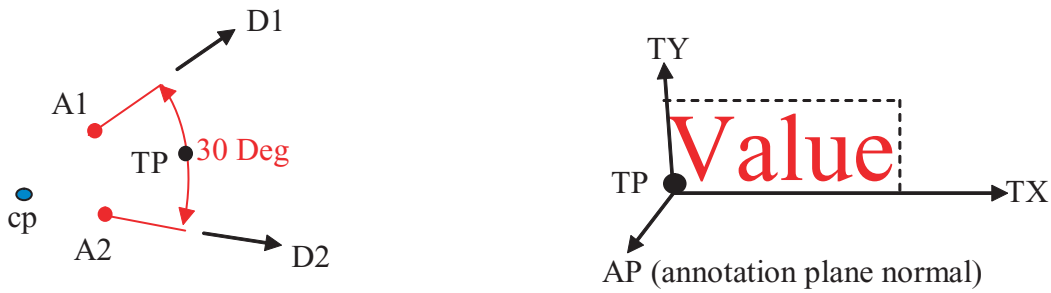

FIGURE 9.23

The angle is defined by the measurement value V (30 in the FIGURE 9.23) and is the angle between the leader direction vectors (D1 and D2). The angular measurement markup is generated by first computing the center point of the angle, cp in this figure.

The text position (TP) controls the position of the measurement text, the placement of the angle arc, and the length and direction of the extension lines. The extension lines are drawn from the anchor point to a point at a distance ||TP-cp|| from the center point cp along the associated direction vector, which is the intersection of the angle arc and the extension line. The angle arc center is at the center point cp (and its radius is ||TP - cp||) and is drawn between the two extensions lines. The markup text is displayed (based on the text orientation parameters) with the lower-left corner of the text string starting at the text position (TP).

The text layout is defined in a similar manner as for other dimensions. The lower-left corner of the text box is positioned at the text anchor point (TP), and the text's X-axis is defined by the vector TX. Note that the vector TX is expected to be orthogonal with the annotation plane normal. The text's up direction is defined as the cross product of the annotation plane normal and the text X-axis, in the direction defined by the TY parameter.

The measurement value is interpreted as either being in degrees or radians as defined by the (DR) value, and the appropriate label string is created.

There are three parts to the text string displayed with the measurement: a numeric value (V), a degrees or radians string (U), and an optional user string (UT). The display of the numeric value field number is also controlled by the precision value (P), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the degrees or radians string and user text as appropriate. This process is viewer dependent.

There are some special cases:

- Parallel direction vectors (D1 and D2) are invalid, and no markup is generated.
- If the text position TP is outside the cone of the angle, an extension line is added to connect the text with the angle arc.

3D Radial Dimension

The *radial measurement* is used to define the radius or diameter of a circular 3D entity. The following figure illustrates two examples of a radial dimension.

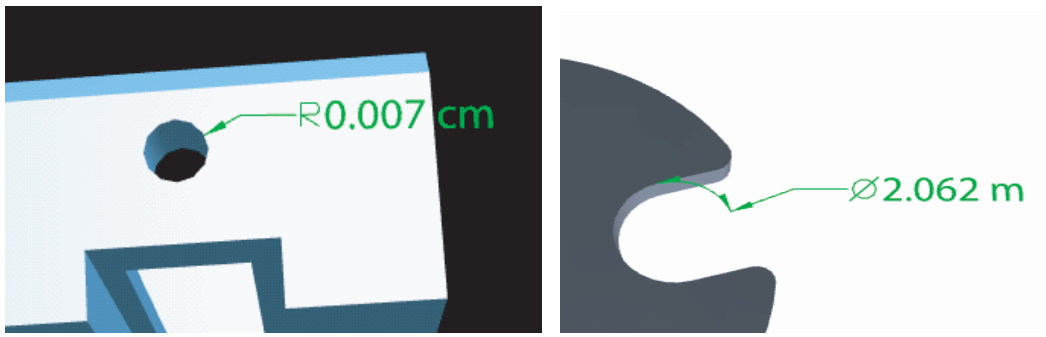

FIGURE 9.24

As shown on the left, the basic markup for a radial dimension consists of an arrow pointing to a circle or arc that is connected to a leader line and text label that defines the radius or diameter. If the arrow is positioned such that it is off the underlying arc, as in the figure on the right, an extension arc is generated that clarifies which arc is being measured.

For radius measurements, the measure value is preceded by an "R" in the measure string. For diameter values, the measure value is preceded by a Greek phi symbol (ϕ).

In addition to the entries in "TABLE 9.39a Entries in a 3D measurement/markup dictionary" on page 62, the following entries are defined for a 3D measurement dictionary with a `Subtype` value of `RD3` for 3D radial measurement.

TABLE 9.39e Additional entries in a 3D measurement/markup dictionary for a 3D radial dimension measurement

KEY	TYPE	VALUE
`AP`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup lies.
`A1`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the first anchor point, called the circle center point, in world space. It is assumed that this is a position on the 3D model associated with this view.
`A2`	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the second anchor point, which is a point on the arc, in world space. It is assumed that this is a position on the 3D model associated with this view.
`N2`	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 2 (`A2`). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.

TABLE 9.39e Additional entries in a 3D measurement/markup dictionary for a 3D radial dimension measurement

KEY	TYPE	VALUE
A3	array	(*Required for arcs only; ExtensionLevel 3*) A three-element array of numbers that defines the start of the arc being measured. This entry is required if the geometry being measured is an arc. See FIGURE 9.26 for a visual representation of A3.
A4	array	(*Required for arcs only; ExtensionLevel 3*) A three-element array of numbers that defines the end of the arc being measured. This entry is required if the geometry being measured is an arc. See FIGURE 9.26 for a visual representation of A4.
TP	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TX	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the horizontal direction vector for the text string presenting the measurement value string.
TY	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the up direction vector for the text string presenting the measurement value string.
EL	number	(*Optional; ExtensionLevel 3*) The length of the extension line in points. The default is 60 points.
TS	number	(*Optional; ExtensionLevel 3*) A number representing the measurement text string height defined in points in the default user space. Note that measurement text is zoom invariant. The default is 12 points.
C	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing the RGB color of the measurement markup. The default value is the array [1 1 1], representing the color white.
V	number	(*Required; ExtensionLevel 3*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
U	text string	(*Required; ExtensionLevel 3*) A string representing the units for the measurement.
P	integer	(*Optional; ExtensionLevel 3*) The number of decimal digits, which represents the precision, shown for the measurement value (V). The default to 3, if P is not specified.
UT	text string	(*Optional; ExtensionLevel 3*) A string defined by the user that is appended to the end of measurement value string. If omitted, no string is appended.

TABLE 9.39e Additional entries in a 3D measurement/markup dictionary for a 3D radial dimension measurement

KEY	TYPE	VALUE
SC	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates whether the underlying circle, or arc, is shown or not. If true, the underlying circle associated with a radial dimension is displayed. The default is false, not to show the circle. **Note:** The circle or arc is redrawn in the markup color (C).
R	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates whether the measurement value is for a radius or a diameter. If true, the measurement value associated with a radial measurement represents a radius, as opposed to a diameter value. The default is true, the measurement value represents a radius.
S	dictionary	(*Optional; ExtensionLevel 3*) A comment reference is an *indirect reference* to a projection annotation that may be associated with this 3D measurement. See "3D Measurements and Projection Annotations" on page 75.

The parameters for defining the radial measurement for a circle are shown here.

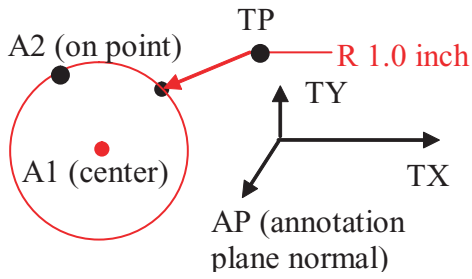

FIGURE 9.25

The circle being measured is defined by two points lying on the annotation plane, a circle center A1, and a point on the circle A2. For radial measurement, the text position (TP) controls the text string position, the arrow line orientation, and the extension line. The arrow line is drawn from the text position (TP) to the intersection point between the circle and a line from text position to the circle center. The extension line will be drawn from the text position (TP) in the direction of the text string's X-axis (TX). The length of the extension line is defined by the EL parameter. The left center of measure value string will begin at the end of the extension line. Note that the vector TX is expected to be orthogonal with the annotation plane normal. The text's up direction is defined as the cross product of the annotation plane normal and the text X-axis, in the direction defined by the TY parameter.

The parameters for defining a radial dimension for an arc are very similar, as shown in FIGURE 9.26.

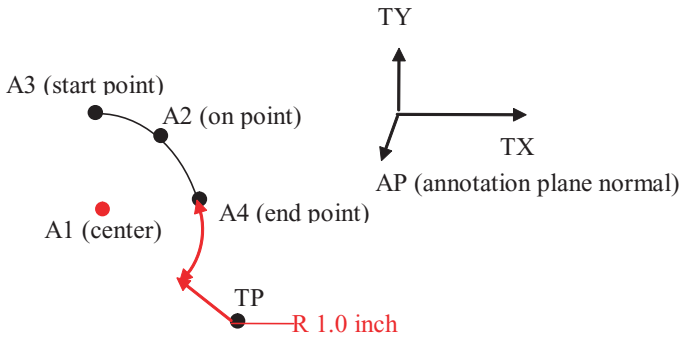

FIGURE 9.26

The key difference for an arc is that there are two addition points defining the start (A3) and end (A4) position for the arc. Additionally for an arc, the text position can be defined such that an extension arc, with an arrowhead on either end, shall be generated, shown in red in FIGURE 9.26.

There are three parts to the text string displayed with the measurement: a numeric value (V), a units string (U), and an optional user string (UT). The display of the numeric value field number is also controlled by the precision value (P), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the units string and user text as appropriate. This process is viewer dependent.

3D Comment Note

3D comment notes let users connect a comment to a specific piece of geometry in the 3D model. The markup consists of a leader line that connects the model to a text box placed in the 3D scene. The text box is rendered so that the text is always facing the user.

Commenting functionality is specified by creating a projection annotation that represents the 3D measurement/markup within the commenting system. See "3D Measurements and Projection Annotations" on page 75 for additional details.

In addition to the entries in "TABLE 9.39a Entries in a 3D measurement/markup dictionary" on page 62, the following entries are defined for a 3D measurement dictionary with a Subtype value of 3DC for 3D comment note.

TABLE 9.39f Additional entries in a 3D measurement/markup dictionary for a 3D comment note

KEY	TYPE	VALUE
A1	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the model space position of the first anchor point in world space. It is assumed that this is a position on the 3D model associated with this view.
N1	text string	(*Optional; ExtensionLevel 3*) The name of the part (or model tree node) associated with anchor point 1 (A1). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.

TABLE 9.39f Additional entries in a 3D measurement/markup dictionary for a 3D comment note

KEY	TYPE	VALUE
TP	array	(*Required; ExtensionLevel 3*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TB	array	(*Optional; ExtensionLevel 3*) A two-element integer array defining the x and y size of the text box to contain the user text string (UT). The x value is defined as a number of characters shown in the x direction (the top row); the y value is the number of rows of text. The conforming reader is free to flow the text in the text box as appropriate. If the entire string does not fit, it may be truncated. The conforming reader may also choose any font in which to render the content, though a monospaced font is recommended.
TS	number	(*Optional; ExtensionLevel 3*) The text size – a number representing the measurement text string height defined in points in the default user space. Note that measurement text is zoom invariant. The default is 12 points.
C	array	(*Optional; ExtensionLevel 3*) An array of three numbers in the range 0.0 to 1.0, representing the Device RGB color of the measurement markup. The default value is the array [1 1 1], representing the color white.
UT	text string	(*Optional; ExtensionLevel 3*) The string displayed as the contents of the 3D comment note.
S	dictionary	(*Optional; ExtensionLevel 3*) A comment reference is an *indirect reference* to a projection annotation that may be associated with this 3D measurement. See "3D Measurements and Projection Annotations" on page 75.

The first anchor point (A1) defines the connection to the model. The model space text position TP field defines placement of the corner of the text box that lies closest to anchor point 1 (A1) in the current view. The user string field contains the text to be fitted into the text box. See implementation note E-10, page 131.

3D Measurements and Projection Annotations

Associations between a 3D measurement and a projection annotation are created by defining an indirect reference to a dictionary in both the projection annotation and the 3D measurement.

Projection annotation: When the projection annotation (see "Projection Annotations" on page 39) is created, the ExData entry (an external data dictionary) is defined and used to manage the association between the 3D measurement and the comment. TABLE 9.39g Entries in the external data dictionary of

a projection annotation describes the entries in the external data dictionary of the projection annotation that are used for referencing a 3D measurement.

3D measurement: The value of the S entry in the 3D measurement dictionary is an indirect reference to a projection annotation dictionary of the associated comment.

TABLE 9.39g Entries in the external data dictionary of a projection annotation

KEY	TYPE	VALUE
Type	name	(*Required; ExtensionLevel 3*) The type of PDF object that this dictionary describes; if present, shall be ExData for an external data dictionary.
Subtype	name	(*Required; ExtensionLevel 3*) The type of external data that this dictionary describes; shall be 3DM for an association to a 3D measurement.
M3DREF	dictionary	(*Required; ExtensionLevel 3*) An indirect reference to a 3D measurement dictionary for which this projection annotation is a comment. See "Projection Annotations" on page 39.

Add the Section 9.6 on Rich Media content.

9.6 Rich Media

Extension level 3 introduces rich media PDF constructs that support playing a SWF file and provide enhanced rich media. With rich media annotation, Flash applications, video, audio, and other multimedia can be attached to a PDF with expanded functionality. It improves upon the existing 3D annotation structure to support multiple multimedia file assets, including Flash video and compatible variations on the H.264 format. The new constructs allow a two-way scripting bridge between Flash and a conforming application. There is support for generalized linking of a Flash application state to a comment or view, which enables video commenting. Finally, actions can be linked to video chapter points.

9.6.1 RichMedia Annotations

The annotation subtype RichMedia shares many low-level structural similarities with the 3D Artwork defined in Section 9.5 of the *PDF Reference*. At the top level, the rich media annotation has two primary custom structures. The RichMediaSettings dictionary is unique to each annotation, but the RichMediaContent dictionary can be shared across rich-media annotations.

The rich media annotation can contain any of the entries of an annotation dictionary. (See implementation note E-12 on page 131.) Table 9.49 shows the additional annotation entries specific to this type of annotation.

TABLE 9.49 Additional entries specific to a RichMedia annotation

KEY	TYPE	VALUE
Subtype	name	(*Required; ExtensionLevel 3*) The type of annotation that the dictionary describes. The name shall be RichMedia for a rich media annotation.
RichMediaSettings	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaSettings dictionary that stores conditions and responses that determine when the annotation should be activated and deactivated and the initial state of artwork in those states. See "RichMediaSettings Dictionary" on page 78. Default value: If no RichMediaSettings dictionary is present, the first configuration is loaded.
RichMediaContent	dictionary	(*Required; ExtensionLevel 3*) A RichMediaContent dictionary that stores the rich media artwork and information as to how it should be configured and viewed. See "RichMediaContent dictionary" on page 86.

As with 3D annotations defined in Section 9.5.1 of the *PDF Reference, sixth edition*, RichMedia annotations can be in one of two states,

Inactive: (the default initial state) The annotation displays the normal appearance of the annotation. Typically, the normal appearance is a 2D bitmap portraying a rendering of the default view of the artwork within the RichMedia annotation.

Active: The annotation displays a rendering of the artwork. This rendering is specified by the annotation's RichMediaSettings entry. (See "TABLE 9.50 Entries in a RichMediaSettings dictionary" on page 78.)

The subsequent sections describe the structural components of a rich media annotation, beginning with the RichMediaSettings dictionary and its subcomponents, followed by the RichMediaContent dictionary, on page 86, and its subcomponents. See the "Extended Example" on page 94 for a detailed and comprehensive example of a rich media annotation.

RichMediaSettings Dictionary

The RichMediaSettings dictionary has a purpose similar to that of the `3DA` dictionary with the 3D Annotation described in Section 9.5.1 of the *PDF Reference*. The RichMediaSettings dictionary stores the conditions and responses that occur in response to certain events, such as activation and deactivation of the annotation, and contains two dictionaries. Table 9.50 gives the details of the content of this dictionary.

TABLE 9.50 Entries in a RichMediaSettings dictionary

KEY	TYPE	VALUE
`Type`	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, shall be `RichMediaSettings` for a RichMediaSettings dictionary.
`Activation`	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaActivation dictionary (see Table 9.50a) that specifies the style of presentation, default script behavior, default view information, and animation style when the annotation is activated.
`Deactivation`	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaDeactivation dictionary (see Table 9.50b) that specifies the condition and type of unloading (restart or pause) that occurs during deactivation.

RichMediaActivation Dictionary

The RichMediaActivation dictionary specifies the style of presentation, default script behavior, default view information, and animation style when the annotation is activated. Table 9.50a details the contents of the dictionary.

The activation dictionary includes a script array. Each element contains an indirect object reference to file specification objects that are also referenced by the `Assets` name tree of the RichMediaContent dictionary (See "RichMediaContent dictionary" on page 86.) When the annotation is activated, each script in the array is executed in order in a common environment per annotation by the rich media run time JavaScript interpreter.

There is an optional reference to a view in the scene's `Views` array (see "TABLE 9.51 Entries in a RichMediaContent dictionary" on page 86) that acts as the default view for the annotation. If not specified,

the first view in the views array is used. If there are no views specified, default values are used for all camera and display parameters.

TABLE 9.50a Entries in a RichMediaActivation dictionary

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, shall be RichMediaActivation for an activation dictionary.
Condition	name	(*Optional; ExtensionLevel 3*) A name that specifies the circumstances under which the annotation should be activated. The following values are valid: • XA, the annotation is explicitly activated by a user action or script. • PO, the annotation is activated as soon as the page that contains the annotation receives focus as the current page. • PV, the annotation is activated as soon as any part of the page that contains the annotation becomes visible. One example is in a multiple-page presentation. Only one page is the current page although several are visible. Default value: XA.
Animation	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaAnimation dictionary that describes the preferred method that conforming readers should use to drive keyframe animations present in this artwork. (See "TABLE 9.50c Entries in a RichMediaAnimation dictionary" on page 81.)
View	dictionary	(*Optional; ExtensionLevel 3*) An indirect object reference to a 3D view dictionary that shall also be referenced by the Views array within the annotation's RichMediaContent dictionary. (See "RichMediaContent dictionary" on page 86.) Default value: The first element in the Views array of the annotation specified in the RichMediaContent dictionary. (See "TABLE 9.51 Entries in a RichMediaContent dictionary" on page 86.) If a Views array does not exist, default values for the components of a 3D view dictionary (see Section 9.5.3 of the *PDF Reference*) are used.
Configuration	dictionary	(*Optional; ExtensionLevel 3*) An indirect object reference to a RichMediaConfiguration dictionary that shall also be referenced by the Configurations array in the RichMediaContent dictionary. (See "TABLE 9.51 Entries in a RichMediaContent dictionary" on page 86.) Default value: The first element within the Configurations array specified in the RichMediaContent dictionary.

TABLE 9.50a Entries in a RichMediaActivation dictionary		
KEY	**TYPE**	**VALUE**
Presentation	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaPresentation dictionary that contains information as to how the annotation and user interface elements will be visually laid out and drawn. See Table 5 for a detailed description.
Scripts	array	(*Optional; ExtensionLevel 3*) An array of indirect object references to file specification dictionaries, each of which describe a JavaScript file that shall be present in the Assets name tree of the RichMediaContent dictionary. Default value: If the array has no elements, no script is executed.

RichMediaDeactivation Dictionary

The second component of the RichMediaSettings dictionary is the RichMediaDeactivation dictionary, which specifies the condition that causes deactivation of the annotation.

TABLE 9.50b Entries in a RichMediaDeactivation dictionary		
KEY	**TYPE**	**VALUE**
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, shall be RichMediaDeactivation for a RichMediaDeactivation dictionary.
Condition	name	(*Optional; ExtensionLevel 3*) A name specifying the circumstances under which the annotation should be deactivated. The following values are valid: • XD, the annotation is explicitly deactivated by a user action or script. • PC, the annotation is deactivated as soon as the page that contains the annotation loses focus as the current page. • PI, the annotation is deactivated as soon as the entire page that contains the annotation is no longer visible. Default value: XD.

RichMediaAnimation Dictionary

A RichMediaAnimation dictionary specifies the preferred method that conforming readers should use to apply timeline scaling to keyframe animations. It can also specify that keyframe animations be played repeatedly. The Animation entry of the RichMediaActivation dictionary (Table 9.50c, page 78) can specify a RichMediaAnimation dictionary.

A keyframe animation can be provided as part of Flash or 3D model content. Keyframe animation is an interactive feature that is highly dependent on the behavior and controls provided by the conforming reader. Table 9.50c shows the entries in a RichMediaAnimation dictionary.

TABLE 9.50c Entries in a RichMediaAnimation dictionary

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, the type shall be RichMediaAnimation for an animation dictionary.
Subtype	name	(*Optional; ExtensionLevel 3*) The animation style described by this dictionary. Valid values are None, Linear, and Oscillating. See Table 9.37 of the *PDF Reference*, sixth edition, PDF 1.7, for descriptions of these animation styles. If an animation style is encountered other than those described, an animation style of None is used. Default value: None
PlayCount	integer	(*Optional; ExtensionLevel 3*) An integer specifying the play count for this animation style. A nonnegative integer represents the number of times the animation is played. A negative integer indicates that the animation is infinitely repeated. This value is ignored for an animation subtype of type None. Default value: -1
Speed	number	(*Optional; ExtensionLevel 3*) A positive number specifying the speed to be used when running the animation. A value greater than one shortens the time it takes to play the animation, or effectively speeds up the animation. This allows authors to change the desired speed of animations without re-authoring the content. This value is ignored for an animation subtype of type None. Default value: 1

Example: *A RichMediaSettings dictionary*

```
20 0 obj                        % RichMediaSettings dictionary
<< /Type /RichMediaSettings
   /Activation
   << /Type /RichMediaActivation % RichMediaActivation Dictionary
      /Condition /XA
      /Configuration 14 0 R     % Reference to element in Configurations array
      /View 19 0 R              % Reference to element in Views array
      /Animation                % RichMediaAnimation dictionary
      << /Type /RichMediaAnimation
         /Subtype /Linear
         /Speed 1
         /PlayCount -1
      >>
      /Presentation 21 0 R
```

```
      /Scripts                % References to scripts in Resources name tree
      [ 32 0 R
        33 0 R
      ]
   >>
   /Deactivation              % RichMediaDeactivation dictionary
   << /Type /RichMediaDeactivation
      /Condition /XD
   >>
>>
endobj
```

RichMediaPresentation Dictionary

The *RichMediaPresentation dictionary* contains information about how the annotation and user interface elements are to be visually laid out and drawn. The visibility of the `Toolbar` is specified, and it will only take effect if the RichMediaConfiguration subtype is `3D`. (See "TABLE 9.51a Entries in a RichMediaConfiguration dictionary" on page 88.)

User interface items such as the navigation pane can be displayed or hidden by default. The navigation pane is a user interface item used to display the hierarchical relationship of entities within the artwork. (See Section 9.5.1, "3D Annotations" of the *PDF Reference, sixth edition*.)

The `Style` of the annotation can be presented `Embedded` within the PDF page or separately `Windowed`. If a `Windowed` state is chosen, the default dimensions and position of the window may be given. Table 9.50d contains a description of each element.

TABLE 9.50d Entries in a RichMediaPresentation dictionary

KEY	TYPE	VALUE
`Type`	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, shall be `RichMediaPresentation` for a presentation dictionary.
`Style`	name	(*Optional; ExtensionLevel 3*) The style of presentation of the rich media. The value can be `Embedded` or `Windowed`. A conforming reader shall support the state `Embedded`. Default value: `Embedded`.
`Window`	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaWindow Dictionary that describes the size and position of the floating user interface window when the value for `Style` is set to `Windowed`. See "RichMediaWindow Dictionary" on page 84 for a detailed description.

TABLE 9.50d Entries in a RichMediaPresentation dictionary

KEY	TYPE	VALUE
Transparent	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates whether the page content is displayed through the transparent areas of the rich media content (where the alpha value is less than 1.0). If true, the rich media artwork is composited over the page content using an alpha channel. If false, the rich media artwork is drawn over an opaque background prior to composition over the page content. (See also implementation note E-13, page 131.) Default value: false
NavigationPane	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates the default behavior of the navigation pane user interface element. If true, the navigation pane is visible when the content is initially activated. If false, the navigation pane is not displayed by default. Default value: false
Toolbar	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates the default behavior of an interactive toolbar associated with this annotation. If true, a toolbar is displayed when the annotation is activated and given focus. If false, a toolbar is not displayed by default. The toolbar should be positioned in proximity to the annotation. Default value: true for content of Subtype 3D, and false otherwise.
PassContextClick	boolean	(*Optional; ExtensionLevel 3*) A flag that indicates whether a context click on the rich media annotation is passed to the media player run time or is handled by the conforming reader. If false, the conforming reader handles the context click. If true, the conforming reader's context menu is not visible, and the user sees the context menu and any custom items generated by the media player run time. Default value: false **Note:** A context click is usually generated by a mouse right-click although it may be invoked by other means. This can include, but is not limited to, an explicit context-menu keyboard key or the combination of a mouse click and a keyboard modifier key.)

RichMediaWindow Dictionary

The RichMediaWindow dictionary stores the dimensions and position of the floating window presented to the user. It is used only if `Style` (RichMediaPresentation dictionary, page 82) is set to `Windowed`. The window described is a non-printing user interface element. A conforming reader constrains the window extent to be within the presentation area of the reader.

All coordinates are expressed in default user space units although do not rotate or scale the window to match the orientation and magnification of the page. The expected behavior is similar to when the flags `NoZoom` and `NoRotate` are set to `true`. (See Section 8.4.2, "Annotation Flags" in *PDF Reference, sixth edition*.) Table 9.50e details the contents of this dictionary.

TABLE 9.50e Entries in a RichMediaWindow dictionary

KEY	TYPE	VALUE
`Type`	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, shall be `RichMediaWindow` for a RichMediaWindow dictionary.
`Width`	dictionary	(*Optional; ExtensionLevel 3*) A dictionary with keys `Default`, `Max`, and `Min` describing values for the width of the Window in default user space units. Default values: `Default` is 288, `Max` is 576, and `Min` is 72
`Height`	dictionary	(*Optional; ExtensionLevel 3*) A dictionary with keys `Default`, `Max`, and `Min` describing values for the width of the Window in default user space units. Default values: `Default` is 216, `Max` is 432, and `Min` is 72
`Position`	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaPosition dictionary describing the position of the RichMediaWindow. See Table 9.50e for a detailed description.

The position of the window in the reader presentation area is described by the RichMediaPosition dictionary. The position of the window remains fixed, regardless of the page translation. Table 9.50f details the contents of this dictionary.

TABLE 9.50f Entries in a RichMediaPosition dictionary

KEY	TYPE	VALUE
`Type`	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, it shall be `RichMediaPosition` for a RichMediaPosition dictionary.
`HAlign`	name	(*Optional; ExtensionLevel 3*) Describes the horizontal alignment. Valid values are `Near`, `Center`, and `Far`. Default value: `Far`

TABLE 9.50f Entries in a RichMediaPosition dictionary

KEY	TYPE	VALUE
VAlign	name	(*Optional; ExtensionLevel 3*) Describes the vertical alignment. Valid values are Near, Center, and Far. Default value: Near
HOffset	number	(*Optional; ExtensionLevel 3*) The offset from the alignment point specified by the HAlign key. A positive value for HOffset, when HAlign is either Near or Center, offsets the position towards the Far direction. A positive value for HOffset, when HAlign is Far, offsets the position towards the Near direction. Default value: 18
VOffset	number	(*Optional; ExtensionLevel 3*) The offset from the alignment point specified by the VAlign key. A positive value for VOffset, when VAlign is either Near or Center, offsets the position towards the Far direction. A positive value for VOffset, when VAlign is Far, offsets the position towards the Near direction. Default value: 18

The descriptions of the values of the HAlign and VAlign entries depend on the reading order. For left-to-right reading languages, as defined by the value of the Lang entry in the document's Catalog, the descriptions follow.

HAlign: Near represents the left edge of the window, and Far represents the right edge of the window.

VAlign: Near represents the top edge of the window, and Far represents the bottom edge of the window.

For right-to-left reading languages, the above descriptions for HAlign are reversed.

Example: *A RichMediaPresentation Dictionary*

```
24 0 obj                              % RichMediaPresentation dictionary
<< /Type /RichMediaPresentation
   /Toolbar /false
   /NavigationPane /false
   /PassContextClick /false
   /Style /Windowed
   /Window 25 0 R
>>
endobj

25 0 obj                              % RichMediaWindow dictionary
<< /Type /RichMediaWindow
   /Height
   << /Default 216
      /Max 432
      /Min 72
```

```
   >>
   /Width
   << /Default 288
      /Max 576
      /Min 72
   >>
   /Position
   << /Type /RichMediaPosition       % RichMediaPosition dictionary
      /HAlign /Far
      /VAlign /Near
      /HOffset 18
      /VOffset 18
   >>
>>
endobj
```

In the above example, the toolbar or the navigation pane are not displayed on initial activation of the annotation. Context click events are handled by the conforming reader and not passed to the annotation handler. The annotation appears as a separate window with an initial width of four inches and an initial height of three inches and will be aligned 0.25 inches off the top right (in left to right reading languages) corner of the document area.

RichMediaContent dictionary

The RichMediaContent dictionary contains content that is present within the annotation as referenced by the RichMediaSettings dictionary. (See "TABLE 9.50 Entries in a RichMediaSettings dictionary" on page 78.) Table 9.51 details the elements of the RichMediaContent dictionary.

TABLE 9.51 Entries in a RichMediaContent dictionary

KEY	TYPE	VALUE
`Type`	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, it shall be `RichMediaContent` for a RichMediaContent dictionary.
`Assets`	name tree	(*Optional; ExtensionLevel 3*) A name tree of embedded file specification dictionaries as detailed in Section 3.10.2 of the *PDF Reference, sixth edition*.
`Configurations`	array	(*Optional; ExtensionLevel 3*) An array where each element is an indirect object reference to a RichMediaConfiguration dictionary.
`Views`	array	(*Optional; ExtensionLevel 3*) An array where each element is an indirect object reference to a 3D view dictionary. See "View Dictionary" on page 92 for the details of the entries of this dictionary. Default value: If no views are specified, default values are used for the components of a view dictionary, including rendering/lighting modes, background color, and camera data.

The Assets name tree

The `Assets` entry takes a name tree of embedded file specification dictionaries. The structure of these name trees is compatible with the `Resources` entry ("TABLE 8.6a Entries in a collection colors dictionary" on page 31) as used by portable collections.

The text string that represents the file name in the name tree shall match the values stored for both the `F` and `UF` keys, which shall be the same value. The file name is encoded as a relative URI and has the following naming restrictions:

- The string shall be a PDF text string.
- The string shall not contain any embedded NULL characters.
- The number of characters in the string shall be between 1 and 255 inclusive.
- The string shall not contain any of these six characters: U+003A COLON (:), U+002A ASTERISK (*), U+0022 QUOTATION MARK ("), U+003C LESS-THAN SIGN (<), U+003E GREATER-THAN SIGN (>), and U+007C VERTICAL LINE (|).
- The last character shall not be a U+002E FULL STOP (.).

The restrictions listed above are inherited from the *Universal Container Format (UCF)* specification. (See the Bibliography.)

Example: *Assets name tree*

```
29 0 obj                        % Assets name tree
<< /Names
   [  (3D.u3d) 30 0 R
      (Flash.swf) 31 0 R
   ]
>>
endobj

30 0 obj                        % File specification dictionary for 3D file
<< /Type /Filespec
   /F (3D.u3d)
   /UF (3D.u3d)
   /EF <</F 40 0 R >>           % Stream containing the 3D file
>>
endobj

31 0 obj                        % File specification dictionary for SWF file
<< /Type /Filespec
   /F (Flash.swf)
   /UF (Flash.swf)
   /EF << /F 41 0 R >>          % Stream containing the Flash file
>>
endobj

40 0 obj                        % Embedded file stream for 3D file
<< /Type /EmbeddedFile          % 3D.u3d
   /Length …
   /Filter …
>>
stream
…Data for 3D.u3d…
```

```
endstream
endobj
```

RichMediaConfiguration Dictionary

The RichMediaConfiguration dictionary describes a set of instances that are loaded for a given scene configuration. The configuration to be loaded when an annotation is activated is referenced by the `Configuration` key in the RichMediaActivation dictionary specified in the RichMediaSettings dictionary. Table 9.51a details the elements of the RichMediaConfiguration dictionary.

TABLE 9.51a Entries in a RichMediaConfiguration dictionary

KEY	TYPE	VALUE
`Type`	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, it shall be `RichMediaConfiguration` for a RichMediaConfiguration dictionary.
`Subtype`	name	(*Optional; ExtensionLevel 3*) A name specifying the primary content type for the configuration. Valid values are `3D`, `Flash`, `Sound`, and `Video`. Default value: If no value is specified, the run time determines the scene type by referring to the type of asset file specified by the first element in the `Instances` array.
`Name`	text tree	(*Optional; ExtensionLevel 3*) A unique name for the configuration.
`Instances`	array	(*Optional; ExtensionLevel 3*) An array of indirect object references to RichMediaInstance dictionaries. (See "TABLE 9.51b Entries in a RichMediaInstance dictionary".)

A RichMediaConfiguration may be an aggregate of several RichMediaInstance dictionaries with different values for subtype. The `Subtype` entry helps to describe the behavior for the collection of those RichMediaInstance dictionaries.

For example, a FLV video file may require a SWF file to view it. Though the primary instance in the configuration may be a SWF file, the annotation is intended for video playback and thus should have a subtype of `Video`.

Setting the `Subtype` suggests the author's intended use of the assets, which better informs the choices when presenting content-specific user interfaces during the authoring or editing process.

RichMediaInstance Dictionary

The RichMediaInstance dictionary, referenced by the `Instances` entry of the RichMediaConfiguration dictionary ("RichMediaConfiguration Dictionary" on page 88), describes a single instance of an asset with settings to populate the artwork of an annotation, as described in Table 9.51b.

TABLE 9.51b Entries in a RichMediaInstance dictionary

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, it shall be RichMediaInstance for a RichMediaInstance dictionary.
Subtype	name	(*Required; ExtensionLevel 3*) A name specifying the content type for the instance. Valid values are 3D, Flash, Sound, and Video. The subtype shall match the asset file type of the instance.
Params	dictionary	(*Optional; ExtensionLevel 3*) A RichMediaParams dictionary as described in "TABLE 9.51c Entries in a RichMediaParams dictionary" on page 90. This entry is valid only for subtype Flash.
Asset	dictionary	(*Required; ExtensionLevel 3*) A dictionary that shall be an indirect object reference to a file specification dictionary that is also referenced in the Assets name tree specified in the RichMediaContent dictionary of the annotation.

Example: *RichMediaInstance dictionaries*

```
15 0 obj                        % RichMediaInstances array
[  16 0 obj
   17 0 obj
]
endobj

16 0 obj                        % RichMediaInstance dictionary
<< /Type /RichMediaInstance
   /Subtype /3D
   /Asset 30 0 R                % Reference to 3D filespec in Assets
>>
endobj

17 0 obj                        % RichMediaInstance dictionary
<< /Type /RichMediaInstance
   /Subtype /Flash
   /Asset 31 0 R                % Reference to Flash filespec in Assets
   /Params 18 0 R
>>
endobj
```

RichMediaParams Dictionary

Contains parameters related to an active Flash subtype in a RichMediaInstance dictionary, as described in Table 9.51c. A RichMediaParams dictionary is not used by content subtypes other than Flash.

TABLE 9.51c Entries in a RichMediaParams dictionary

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, it shall be RichMediaParams for a RichMediaParams dictionary.
FlashVars	text string or stream	(*Optional; ExtensionLevel 3*) A text string or stream containing formatted name value pairs passed to the Flash Player context when activated. For the format specifics, see the document titled *Using FlashVars to pass variables to a SWF,* listed in the Bibliography. Default value: No data is sent to the Flash Player.
Binding	name	(*Optional; ExtensionLevel 3*) Values can be None, Foreground, Background, or Material. The descriptions for these values are given in Table 9.51d, page 88. Default value: None
BindingMaterialName	text string	(*Required if Binding value is Material; ExtensionLevel 3*) Stores the material name that content is to be bound to.
CuePoints	array	(*Optional; ExtensionLevel 3*) An array of CuePoint dictionaries containing points in time within a Flash animation. See "CuePoint Dictionary" on page 91 for a detailed description. Default values: an empty array
Settings	text string or stream	A text string/stream used to store settings information associated with a Flash RichMediaInstance. It is to be stored and loaded by the scripting run time. See the discussion of the Settings entry that follows for more information. Default value: undefined

Table 9.51d contains descriptions of the values of the Binding key of Table 9.51c.

TABLE 9.51d Definitions of the Binding name values

KEY	DESCRIPTION
None	The Flash content is unbound and is not visible at playback time.
Foreground	The Flash content is bound to the foreground and is rendered in front of the 3D model and background Flash content in the active annotation. If more than one RichMediaInstance has a RichMediaParams dictionary Binding value of Foreground, the Flash content is rendered in order from back to front, each instance composited over prior instances using its alpha channel.
Background	The Flash content is bound to the background and is rendered behind any 3D model content or Flash foreground elements in the active annotation. For a given RichMedia annotation, there can be one active RichMediaInstance that has a RichMediaParams dictionary Binding value of Background. If more than one is specified, the last RichMediaInstance specified for the background is used.
Material	The Flash content is bound to a material that is part of 3D content. If that material is applied to geometry within a 3D scene, the Flash appears to be playing upon this object as if conforming to the surface of the object.

The Settings entry (as described in Table 9.51c) in a RichMediaParams dictionary is used to store and load settings information specific to the content referenced by the resource in the same RichMediaInstance dictionary. The value of the Settings entry is passed by the ActionScript ExternalInterface command multimedia_loadSettingsString, where the value of Settings is passed as the first and only argument.

The Settings string is updated when the Flash content sends the following ActionScript command to the player:

```
ExternalInterface.call( "multimedia_saveSettingsString", settings );
```

In this command, settings is the argument and is the new string value to be written to the Settings entry. This value is honored only when the conforming reader has the rights to edit the document, and results in the document being dirtied.

CuePoint Dictionary

A video file can contain cue points that are encoded in a video stream or may be created by an associated ActionScript within the Flash content. The CuePoint dictionary, as described in Table 9.51e, contains a state that relates the cue points to an action that may be passed to the conforming application or may be used to change the appearance. Cue points in the Flash content are matched to the cue points declared in the PDF file by the values specified by the Name or Time keys.

TABLE 9.51e Entries in a CuePoint dictionary

KEY	TYPE	VALUE
Type	name	(*Optional; ExtensionLevel 3*) The type of PDF object that this dictionary describes. If present, it shall be CuePoint for a CuePoint dictionary.
Subtype	name	(*Optional; ExtensionLevel 3*) Values can be Navigation or Event, as described here: A Navigation cue point is an event encoded in a Flash movie (FLV). A chapter stop may be encoded so that when the user requests to go to or skip a chapter, a navigation cue point is used to indicate the location of the chapter. An Event is a generic cue point of no specific significance other than a corresponding action is triggered.
Name	text string	(*Required; ExtensionLevel 3*) The name of the cue point to match against the cue point within Flash content and for display purposes.
Time	number	(*Required; ExtensionLevel 3*) The time value of the cue point in milliseconds to match against the cue point within Flash content and for display purposes.
A	dictionary	(*Required; ExtensionLevel 3*) An action dictionary defining the action that is executed if this cue point is triggered, meaning that the Flash content reached the matching cue point during its playback. (See Section 8.5.1 of the *PDF Reference*, *sixth edition*.)

Example: *The RichMediaParams Dictionary*

```
18 0 obj                        % RichMediaParams dictionary
<< /Type /RichMediaParams
   /Binding /Foreground
   /FlashVars (name=John+Smith&address=1+Main+St.&city=Springfield)
   /CuePoints
   [
      << /Type /CuePoint
         /Name (Cue Point 1)
         /Subtype /Navigation
         /Time 5100
      >>
      << /Name (Cue Point 2)
         /Subtype /Event
         /Time 4020
         /A 19 0 R
      >>
   ]
>>
endobj
```

View Dictionary

View dictionaries specify a unique view of the artwork that can be used when the annotation is activated or selected. A View dictionary entry in a rich media annotation contains a superset of the information

present in a 3D view (3DV) dictionary as specified in Section 9.5.3 of the *PDF Reference*. Table 9.51f describes the entries in addition to those present within a 3D view dictionary.

TABLE 9.51f Additional entries to a 3D view dictionary

KEY	TYPE	VALUE
Snapshot	stream	(*Optional; ExtensionLevel 3*) A stream that contains an image XObject (see Section 4.8, "Images") to be displayed when the view is invoked. Default value: No image is used, the currently active rich media content is displayed.
Params	array	(*Optional; ExtensionLevel 3*) An array containing View Params dictionaries. See TABLE 9.51g Entries in a View Params dictionary for a detailed description. Default value: An empty array

If the Snapshot entry exists, the image it describes is displayed in place of the current artwork. Because the snapshot image is integrated by the player with other elements, such as user interface controls for video playback, the color space of the image may be modified.

View Params Dictionary

The View Params dictionary provides a reference to a RichMediaInstance dictionary within the Instances array, and additional Data that is passed to the instance. This data may be formatted to serve the individual purpose of the content being referenced.

TABLE 9.51g Entries in a View Params dictionary

KEY	TYPE	VALUE
Instance	dictionary	(*Required; ExtensionLevel 3*) A dictionary that shall be an indirect object reference to a RichMediaInstance dictionary that is also present in the Instances array of the annotation.
Data	text string or stream	(*Required; ExtensionLevel 3*) A text string or stream containing state data to be passed to the instance when the view is triggered.

The Data entry (as described in Table 9.51g) in a View Params dictionary is used to store and load opaque data specific to the Instance referenced in the same dictionary when a View is activated. This system allows content-specific state information to be stored when a view is created, usually during a comment or drawing markup.

Saving State Data. When the View is created, the run time iterates through all active instances for the current configuration and queries them for state data. The mechanism by which the content of the Data entry is retrieved from the content instance depends on the content type:

Flash content: An ActionScript ExternalInterface call is made to the function multimedia_getState with no arguments. The return value, if it is present and can be interpreted as a string, is stored as the value for Data.

3D content: A JavaScript StateEvent is invoked with the type property set to the value save. After the event is handled, the value for the data property, if it is present and can be interpreted as a string, is stored as the value for Data.

This is honored only when the conforming reader has the usage rights to edit the document. It results in the document being dirtied.

Loading State Data. When the View is activated, the run time iterates through each View Params dictionary and loads the state data for each `Instance` referenced. The mechanism by which the content of the `Data` entry is restored depends on the content type:

Flash content: An ActionScript ExternalInterface call is made to the function `multimedia_setState`, where the first argument is the value stored for `Data`.

3D content: A JavaScript `StateEvent` is invoked with the `type` property set to the value `load` and the `data` property set to the value stored for `Data`.

See implementation note E-11, page 131.

State Data for Commenting on Video. RichMedia annotations that support video playback have a Flash (.swf) resource to enable playback of the video content. That file needs to support saving and loading state data in order to enable commenting on Video.

The format of the state data when a `multimedia_getState` ExternalInterface command is invoked is as follows, where *currentTimeInSeconds* is the current time the video playhead is set at:

`'<stateData><movieInfo time="`*currentTimeInSeconds*`"/></stateData>'`

Sending a `multimedia_setState` ExternalInterface command to the video player Flash instance with the above string as an argument sets the current time to match the value for *currentTimeInSeconds*.

Extended Example

The following is a quasi complete example of a rich media annotation.

```
9 0 obj                          % RichMedia annotation
<< /Type /Annot
   /Subtype /RichMedia
   /NM (RichMedia001)            % Annotation name
   /AP << /N 10 0 R >>           % Appearance dictionary
   /F 68                         % Annotation flags
   /P 5 0 R                      % Parent
   /Rect [ 50 50 742 500 ]       % Rectangle
   /Border [ 0 0 0 ]             % Border
   /BS                           % Border Style dictionary
   << /Type /Border
      /W 0                       % Width (0 points)
      /S /S                      % Border style (Solid)
   >>
   /RichMediaContent 12 0 R
   /RichMediaSettings 22 0 R
>>
endobj

10 0 obj                         % Appearance dictionary
<< /Length 39
   /Type /XObject
   /BBox [ 0.0 0.0 692 450 ]
   /Resources
   << /XObject << /Im0 11 0 R >>
      /ProcSet [ /PDF /ImageC ]
```

```
      /ExtGState <</GS0 << /Type /ExtGState /ca 1.0 /CA 1.0 >> >>
   >>
   /Subtype /Form
   /Matrix [ 1 0 0 1 0 0 ]
>>
stream
q
/GS0 gs
692 0 0 450 0 0 cm
/Im0 Do
Q
endstream
endobj

11 0 obj
<< /Length ...
   /Width ...
   /Height ...
   /BitsPerComponent 8
   /ColorSpace /DeviceRGB
   /Subtype /Image
>>
stream
...Poster Image Stream...
endobj

12 0 obj                        % RichMediaContent dictionary
<< /Type /RichMediaContent
   /Configurations 13 0 R
   /Views 15 0 R
   /Assets 29 0 R
>>
endobj

13 0 obj                        % RichMediaConfigurations array
[ 14 0 R ]
endobj

14 0 obj                        % RichMediaConfiguration dictionary
<< /Type /RichMediaConfiguration
   /Name (Configuration01)
   /Instances 15 0 R
>>
endobj

15 0 obj                        % RichMediaInstances array
[ 16 0 obj
  17 0 obj
]
endobj

16 0 obj                        % RichMediaInstance dictionary
<< /Type /RichMediaInstance
   /Subtype /3D
   /Asset 30 0 R                % Reference to 3D filespec in Resources
```

```
>>
endobj

17 0 obj                        % RichMediaInstance dictionary
<< /Type /RichMediaInstance
   /Subtype /Flash
   /Asset 31 0 R
   /Params 18 0 R
>>
endobj

18 0 obj                        % Params dictionary
<< /Type /RichMediaParams
   /Binding /Foreground
   /FlashVars (name=John+Smith&address=1+Main+St.&city=Springfield)
   /CuePoints
   [
      << /Type /CuePoint
         /Name (Cue Point 1)
         /Subtype /Nav
         /Time 5100
      >>
      << /Type /CuePoint
         /Name (Cue Point 2)
         /Subtype /Event
         /Time 4020
         /A 19 0 R
      >>
   ]
>>
endobj

19 0 obj                        % JavaScript action dictionary
<< /Type /Action
   /S /JavaScript
   /JS (app.alert\("Cue Point Reached"\);)
   /Next 20 0 R
>>
endobj

20 0 obj                        % RichMediaExecute action
<< /Type /Action
   /S /RichMediaExecute
   /TA 9 0 R
   /CMD
   << /A (Argument String)
      /C (CommandName)
   >>
>>
endobj

21 0 obj                        % Views array
[ 22 0 R ]
endobj
```

```
22 0 obj                        % 3DView dictionary
<< /Type /3DView
   /IN (12a345b6-7c89-0d1e-fa2b-cde3f45a6b78)
   /XN (Default View)
   /CO 10.0
   /MS /M
   /NR true
   /RM << /Type /3DRenderMode /Subtype /Solid >>
   /P << /Subtype /P /PS /Min /FOV 30.0 >>
   /BG <</Subtype /SC /C [ .5 .5 .5 ] >>
   /C2W [ -1.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 -10.0 0 ]
   << /Instance 16 0 R
      /Data (...State Data for 3D Instance...)
   >>
>>
endobj

23 0 obj                        % RichMediaSettings Dictionary
<< /Type /RichMediaSettings
   /Activation
   << /Type /RichMediaActivation
      /Condition /WhenClicked
      /Configuration 14 0 R     % Reference to element in Configurations array
      /View 22 0 R              % Reference to element in Views array
      /Animation                % RichMediaAnimation dictionary
      << /Type /RichMediaAnimation
         /Subtype /Linear
         /Speed 1
         /PlayCount -1
      >>
      /Presentation 24 0 R
      /Scripts                  % Reference to JavaScript filespec in Resources
      [ 34 0 R
        35 0 R
      ]
   >>
   /Deactivation
   << /Type /RichMediaDeactivation
      /Condition /XD
   >>
>>
endobj

25 0 obj                        % RichMediaWindow dictionary
<< /Type /RichMediaWindow
   /Height
   << /Default 216
      /Max 432
      /Min 72
   >>
   /Width
   << /Default 288
      /Max 576
      /Min 72
   >>
```

```
   /Position
   << /Type/RichMediaPosition % RichMediaPosition dictionary
      /HAlign /Far
      /VAlign /Near
      /HOffset 18
      /VOffset 18
   >>
>>
endobj

29 0 obj                      % RichMediaAssets name tree
<< /Names
   [  (3D.u3d) 30 0 R
      (Player.swf) 31 0 R
      (Controls.swf) 32 0 R
      (Video.flv) 33 0 R
      (Script.js) 34 0 R
      (Navigation.js) 35 0 R
   ]
>>
endobj

30 0 obj                      % File specification dictionary for 3D file
<< /Type /Filespec
   /F (3D.u3d)
   /UF (3D.u3d)
   /EF <</F 40 0 R >>         % Stream containing the 3D file
>>
endobj

31 0 obj                      % File specification dictionary for Flash file
<< /Type /Filespec
   /F (Flash.swf)
   /UF (Flash.swf)
   /EF << /F 41 0 R >>        % Stream containing the Flash file
>>
endobj

34 0 obj                      % File specification dict. for JavaScript file
<< /Type /Filespec
   /F (Script.js)
   /UF (Script.js)
   /EF << /F 44 0 R >>        % Stream containing the JavaScript file
>>
endobj

35 0 obj                      % File specification dict. for JavaScript file
<< /Type /Filespec
   /F (Navigation.js)
   /UF (Navigation.js)
   /EF << /F 45 0 R >>        % Stream containing the JavaScript file
>>
endobj

40 0 obj                      % Embedded file stream for 3D file
```

```
<< /Type /EmbeddedFile        % 3D.u3d
   /Length ...
   /Filter ...
>>
stream
...Data for 3D.u3d...
endstream
endobj

41 0 obj                      % Embedded file stream for Flash file
<< /Type /EmbeddedFile        % Flash.swf
   /Length ...
   /Filter ...
>>
stream
...Data for Flash.swf...
endstream
endobj

44 0 obj                      % Embedded file stream for JavaScript file
<< /Type /EmbeddedFile        % Script.js
   /Length ...
   /Filter ...
>>
stream
...Data for Script.js...
endstream

endobj
45 0 obj                      % Embedded file stream for JavaScript file
<< /Type /EmbeddedFile        % Navigation.js
   /Length ...
   /Filter ...
>>
stream
...Data for Navigation.js...
endstream
endobj
...
```

Document Interchange (Chapter 10 in PDF Reference)

This section describes extensions to the PDF specification contained in the *PDF Reference, sixth edition, version 1.7* and in ISO 32000 (*Document management - Portable document format - PDF 1.7*). The latter document is the version of the PDF specification that has been ratified by the ISO. These documents differ as described in *About the ISO Draft of the PDF 1.7 Reference*.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

10.7 Tagged PDF

10.7.1 Tagged PDF and Page Content

Real Content and Artifacts

On page 886, add the new value `BatesN` *to the list of values supported by* `Subtype` *and add the new* `Contents` *dictionary entry, as shown in Table 10.17 that follows. Unchanged content appears in gray.*

TABLE 10.17 Property list entries for artifacts

KEY	SUBTYPE	DESCRIPTION
`Subtype`	name	(*Optional; PDF 1.7*) The subtype of the artifact. This entry applies only when the `Type` entry has a value of `Pagination`. Valid values are `Header`, `Footer`, and `Watermark`. Additional values can be defined for this entry, provided they comply with the naming conventions described in Appendix E. ExtensionLevel 3 adds `BatesN` to the set of valid values.
`Contents`	text string or array of text string	(*Optional; ExtensionLevel 3*) The artifact content, which applies when `Subtype` has a value of `BatesN`. If this artifact applies to a page that contains a single Bates Number, the `Contents` value is a Bates Numbering string. If this artifact applies to a page that contains multiple Bates Numbers, the `Contents` value is an array of text strings, each of which contains a concatenation of the Bates Numbering parts. The order in which the text strings appear in the array is application-dependent. **Note:** Typically, applications that support Bates Numbering add private data to the PDF document that specifies starting values for Bates Numbering, and placement and formatting of those values. The intent of Bates Numbering values provided in this artifact content is to override Bates Numbering values in the private data.

Replace the bullet that begins "Pagination artifacts" with the following bullet. Unchanged content appears in gray.

- Pagination artifacts. Ancillary page features such as running heads, folios (page numbers), and Bates Numbering. Bates Numbering is a method of indexing legal documents for easy identification and retrieval.

Bibliography

This section describes extensions to the PDF specification contained in the *PDF Reference, sixth edition, version 1.7* and in ISO 32000 (*Document management - Portable document format - PDF 1.7*). The latter document is the version of the PDF specification that has been ratified by the ISO. These documents differ as described in *About the ISO Draft of the PDF 1.7 Reference*.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

Resources from Adobe Systems Incorporated

Acrobat ActionScript API Reference, available at <http://www.adobe.com/go/acrobat_developer>

Acrobat Navigator File Format, available at <http://www.adobe.com/go/acrobat_developer>

PRC specification: The following documents collectively provide the PRC file format specification. These documents are available at www.adobe.com/go/acrobat3d_developer:

- *Adobe PRC Format Specification, version 7298, Adobe Extension Level 1, Adobe Systems Incorporated.*
- *Geometry Compression Level 2*
- *Tesselation Compression Level 2*

Add the new entry shown below. Unchanged text appears in gray.

Adobe XML Architecture specifications, available at <www.adobe.com/go/xfa_ref_24_bibliography>.

Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 2.6

Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 2.5

Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 2.4

Adobe Adobe XML Architecture, Forms Architecture (XFA) Specification, version 2.2

Note: Beginning with XFA 2.2, the XFA specification includes the Template Specification, the Config Specification, the XDP Specification, and all other XML specifications unique to the XML Forms Architecture (XFA).

Adobe XML Architecture, XML Data Package (XDP) Specification, version 2.0

Adobe XML Architecture, Template Specification, version 2.0

Adobe XML Architecture, XML Forms Data Format Specification, version 2.0 (Draft)

Flex 2 Developer Guide, available at <http://www.adobe.com/go/flex_devcenter>

Navigator Template Format, available at <http://www.adobe.com/go/navigator_developer>

Universal Container Format (UCF), available at <http://www.adobe.com/go/acrobat_developer>

Using FlashVars to pass variables to a SWF, TechNote tn_16417, available at <http://www.adobe.com/go/tn_16417>

Other Resources

Replace the following Ecma reference. This change reflects a correction documented in the Errata for the PDF Reference, sixth edition, version 1.7.

Ecma International, Standard ECMA-363, *Universal 3D File Format, 1st Edition*. This document is available at www.ecma-international.org.

Replace this Ecma reference with this:

Ecma International. The following standards are available through <www.ecma-international.org.>.

- ECMA-363, *Universal 3D File Format, 1st Edition*.
- ECMA-363, *Universal 3D File Format, 3rd Edition*.

Add the following bulleted references to the list of Federal Information Processing Standards Publications:

Federal Information Processing Standards Publications:

- FIPS PUB 140-2: *Security Requirements For Cryptographic Modules* <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>

Add the following bullet to the ISO listing of standards:

International Organization for Standardization (ISO). The following standards are available through < http://www.iso.org/>:

- *Information technology – Automatic identification and data capture techniques – Data Matrix bar code symbology specification*, ISO/IEC 16022
- *Information technology –Automatic identification and data capture techniques – PDF417 bar code symbology specification*, ISO/IEC 15438
- *Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification*, ISO/IEC 18004

Add these bulleted references to the list of RFCs:

Internet Engineering Task Force (IETF) Requests for Comments (RFCs). The following RFCs are available through < http://www.rfc-editor.org>:

- RFC 3454, *Preparation of Internationalized Strings ("stringprep")*
- RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*
- RFC 3987, *Internationalized Resource Identifiers (IRIs)*
- RFC 4013, *SASLprep: Stringprep Profile for User Names and Passwords*
- RFC 4122, *A Universally Unique Identifier (UUID) URN Namespace*

Open Geopatial Consortium Documents

- *OpenGIS© Implementation Specification: Coordinate Transformation Services*, document 01-009 <http://www.opengeospatial.org>

Unicode Consortium publications:

- UTS 35, *Unicode Technical Standard #35 Locale Data Markup Language (LDML)*

Part II: Reference Errors and Implementation Notes

Implementation Notes

This chapter provides notes on the implementation of Acrobat 9. The notes are listed in the order of the sections to which they refer in the main text. There are two parts to the implementation notes:

Implementation Notes to the PDF Reference, sixth edition: These notes appear in Appendix H, "Compatibility and Implementation Notes" of *PDF Reference, sixth edition, version 1.7*. Two of the notes, notes 77 and 156, are significantly modified.

Implementation Notes to Adobe Extensions to ISO 32000: These notes remark on Acrobat's implementation of the extensions to the PDF language, as documented in Part I, "Extensions to the PDF Specification", of this guide. The implementation notes for the Adobe extensions begin on page 130.

Note: This guide uses italics to identify information not intended for inclusion in the PDF specification. For convenience, the phrase *BaseVersion 1.7, Adobe ExtensionLevel 3* is shortened to *ExtensionLevel 3* throughout this document.

Implementation Notes to the PDF Reference, sixth edition

These notes originally appeared in the *PDF Reference*. See that document for the context of each note.

1.2 Introduction to PDF 1.7 Features

This note applies to the first paragraph in this section.

1. The native file formats of Acrobat products are PDF 1.2 for Acrobat 3.0, PDF 1.3 for Acrobat 4.0, PDF 1.4 for Acrobat 5.0, PDF 1.5 for Acrobat 6.0, PDF 1.6 for Acrobat 7.0, and PDF 1.7 for Acrobat 8.0.

   Beginning with Acrobat 8.1, the native file formats for Acrobat products are extensions relative to a base version of PDF. The extensions are described in the *Adobe Supplement to ISO 32000* (this document). For base version 1.7, the following is a list of each Adobe extension level and the applications (shown in parentheses) supported by that extension level.

   - ISO 32000: XFA 2.5 and PDF 1.7 (Acrobat 8.0, LiveCycle, SAP P3B)
   - Adobe Extension Level 1: PRC and XFA 2.6 (Acrobat 8.1, LiveCycle ES), the addition of PRC for 3D data.
   - Adobe Extension Level 2: XFA 2.7 (SAP P3C)
   - Adobe Extension Level 3: XFA 2.8 (Acrobat 9.0, LiveCycle ES 8.2) and all the information contained in this document concerning additions to the PDF language.

3.1.2 Comments

2. Acrobat viewers do not preserve comments when saving a file.

3.2.4 Name Objects

3. In PDF 1.1, the number sign character (#) could be used as part of a name (for example, /A#B), and the specifications did not specifically prohibit embedded spaces (although Adobe producer applications did not provide a way to write names containing them). In PDF 1.2, the number sign became an escape

character, preceding two hexadecimal digits. Thus, a 3-character name A-space-B can now be written as /A#20B (since 20 is the hexadecimal code for the space character). This means that the name /A#B is no longer valid, since the number sign is not followed by two hexadecimal digits. A name object with this value must be written as /A#23B, since 23 is the hexadecimal code for the character #.

4. In cases where a PostScript name must be preserved or where a string is permitted in PostScript but not in PDF, the Acrobat Distiller application uses the # convention as necessary. When an Acrobat viewer generates PostScript, it inverts the convention by writing a string where permitted or a name otherwise. For example, if the string (Adobe Green) were used as a key in a dictionary, Distiller would use the name /Adobe#20Green and the viewer would generate (Adobe Green).

5. In Acrobat 4.0 and earlier versions, a name object being treated as text is typically interpreted in a host platform encoding, which depends on the operating system and the local language. For Asian languages, this encoding may be something like Shift-JIS or Big Five. Consequently, it is necessary to distinguish between names encoded this way and ones encoded as UTF-8. Fortunately, UTF-8 encoding is very stylized and its use can usually be recognized. A name that does not conform to UTF-8 encoding rules can instead be interpreted according to host platform encoding.

3.2.7 Stream Objects

The following note is associated with the description of the `DecodeParms` *key in Table 3.4.*

7. Acrobat viewers accept the name `DP` as an abbreviation for the `DecodeParms` key in any stream dictionary. If both `DP` and `DecodeParms` entries are present, `DecodeParms` takes precedence.

3.2.9 Indirect Objects

8. Acrobat viewers require that the name object used as a key in a dictionary entry be a direct object; an indirect object reference to a name is not accepted.

3.3 Filters

9. Acrobat viewers accept the abbreviated filter names shown in table titled “Abbreviations for standard filter names” in addition to the standard ones. These abbreviated names are intended for use only in the context of inline images (see Section 4.8.6, “Inline Images”), they should not be used as filter names in any stream object.

Abbreviations for standard filter names

STANDARD FILTER NAME	ABBREVIATION
`ASCIIHexDecode`	`AHx`
`ASCII85Decode`	`A85`
`LZWDecode`	`LZW`
`FlateDecode` (PDF 1.2)	`Fl` (uppercase F, lowercase L)
`RunLengthDecode`	`RL`

Abbreviations for standard filter names

STANDARD FILTER NAME	ABBREVIATION
`CCITTFaxDecode`	`CCF`
`DCTDecode`	`DCT`

10. If an unrecognized filter is encountered, Acrobat viewers report the context in which the filter was found. If errors occur while a page is being displayed, only the first error is reported. The subsequent behavior depends on the context, as described in table titled "Acrobat behavior with unknown filter". Acrobat operations that process pages, such as the Find command and the Create Thumbnails command, stop as soon as an error occurs.

If an unrecognized filter is encountered, Acrobat viewers report the context in which the filter was found. If errors occur while a page is being displayed, only the first error is reported. The subsequent behavior depends on the context, as described in the table titled "Acrobat behavior with unknown filter". Acrobat operations that process pages, such as the Find command and the Create Thumbnails command, stop as soon as an error occurs.

Acrobat behavior with unknown filter

CONTEXT	BEHAVIOR
Content stream	Page processing stops.
`Indexed color space`	The image does not appear, but page processing continues.
Inline image	The image does not appear, but page processing continues.
Thumbnail image	An error is reported and no more thumbnail images are displayed, but the thumbnails can be deleted and created again.
Form XObject	The form does not appear, but page processing continues.
Type 3 glyph description	The glyph does not appear, but page processing continues. The text position is adjusted based on the glyph width.
Embedded font	The viewer behaves as if the font is not embedded.

3.3.7 DCTDecode Filter

11. Acrobat 4.0 and later viewers do not support the combination of the `DCTDecode` filter with any other filter if the encoded data uses the progressive format. If a version of the Acrobat viewer earlier than 4.0 encounters `DCTDecode` data encoded in progressive format, an error occurs that is handled according to the table "Acrobat behavior with unknown filter" on page 108.

3.4 File Structure

12. Acrobat viewers do not enforce the restriction on line length.

3.4.1 File Header

13. Acrobat viewers require only that the header appear somewhere within the first 1024 bytes of the file.

14. Acrobat viewers also accept a header of the form

`%!PS-Adobe-`*`N.n`*` PDF-`*`M.m`*

3.4.3 Cross-Reference Table

15. Acrobat viewers do not enforce the restriction on object numbers existing in more than one subsection; they use the entry in the first subsection where the object number is encountered. However, overlap is explicitly prohibited in cross-reference streams in PDF 1.5.

16. Acrobat 6.0 and later do not use the free list to recycle object numbers; new objects are assigned new numbers.

17. Acrobat viewers do not raise an error in cases where there are gaps in the sequence of object numbers between cross-reference subsections. The missing object numbers are treated as free objects.

3.4.4 File Trailer

18. Acrobat viewers require only that the `%%EOF` marker appear somewhere within the last 1024 bytes of the file.

3.4.6 Object Streams

19. When creating or saving PDF files, Acrobat 6.0 limits the number of objects in individual object streams to 100 for linearized files and 200 for non-linearized files.

3.4.7 Cross-Reference Streams (Cross-Reference Stream Dictionary)

20. `FlateDecode` is the only filter supported by Acrobat 6.0 and later viewers for cross-reference streams. These viewers also support unencoded cross-reference streams.

3.4.7 Cross-Reference Streams (Compatibility with PDF 1.4)

21. Byte addresses can be as large as needed to address an arbitrarily large PDF file, regardless of the implementation limit for PDF integers in general.

3.5 Encryption

22. An option to use an unpublished algorithm was needed because of an export requirement of the U.S. Department of Commerce. This requirement no longer exists. Acrobat 7.0 does not use this algorithm to encrypt documents, although it can decrypt files that are encrypted with the algorithm.

23. Acrobat viewers will fail to open a document encrypted with a `V` value defined in a version of PDF that the viewer does not understand.

24. Security handlers are responsible for protecting the value of `EFF` from tampering if needed. Acrobat security handlers do not provide this protection.

3.5.2 Standard Security Handler (Standard Encryption Dictionary)

25. Acrobat viewers implement this limited mode of printing as “Print As Image,” except on UNIX systems, where this feature is not available.

3.5.2 Standard Security Handler (Encryption Key Algorithm)

26. The first element of the `ID` array generally remains the same for a given document. However, in some situations, Acrobat may regenerate the `ID` array if a new generation of a document is created. Security handlers are encouraged not to rely on the `ID` in the encryption key computation.

3.5.2 Standard Security Handler (Password Algorithms)

27. In Acrobat 2.0 and 2.1 viewers, the standard security handler uses the empty string if there is no owner password in step 1 of Algorithm 3.3.

3.5.4 Crypt Filters

28. In Acrobat 6.0, crypt filter usage is limited to allowing document-level metadata streams to be left as plaintext in an otherwise encrypted document. In Acrobat 7.0, crypt filter usage also includes the ability to encrypt embedded files while leaving the remainder of the document as plaintext.

29. In Acrobat 6.0 and later, when strings and streams in an encrypted document are edited, those streams and strings are encrypted with the `StmF` and `StrF` filters, respectively. In Acrobat 7.0, if the `EFF` entry in the encryption dictionary is set, embedded files are encrypted with the crypt filter specified by the `EFF` entry. In both Acrobat 6.0 and 7.0, if the security handler indicates that document-level metadata is to be in plaintext, the metadata will not be encrypted. It is up to individual security handlers to store their own flags that indicate whether document-level metadata should be in plaintext.

30. Acrobat viewers do not support the ability for third-party security handlers to specify `None` as a value for `CFM`.

31. In the file specification dictionary (see Section 3.10.2, “File Specification Dictionaries”), related files (`RF`) must use the same crypt filter as the embedded file (`EF`).

32. The value of the `EncryptMetadata` entry is set by the security handler rather than the Acrobat.

3.6.1 Document Catalog

33. .Acrobat 5.0 and Acrobat 6.0 avoid adding a `Version` entry to the document catalog and do so only if necessary. Once they have added this entry, they behave in this way:
    - Acrobat 5.0 never removes the `Version` entry. For documents containing a `Version` entry, Acrobat 5.0 attempts to ensure that the version specified in the header matches the version specified in the `Version` entry; if this is not possible, it at least ensures that the latter is later than (and therefore overrides) the version specified in the header.
    - Acrobat 6.0 removes the `Version` entry when doing a full (non-incremental) save of the document.

34. In PDF 1.2, an additional entry in the document catalog, named `AA`, was defined but was never implemented. The `AA` entry that is newly introduced in PDF 1.4 is entirely different from the one that was contemplated for PDF 1.2.

3.6.2 Page Tree (Page Objects)

35. In PDF 1.2, an additional entry in the page object, named `Hid`, was defined but was never implemented. Beginning with PDF 1.3, this entry is obsolete and should be ignored.

36. Acrobat 5.0 and later viewers do not accept a `Contents` array containing no elements.

37. In a document containing articles, if the first page that has an article bead does not have a `B` entry, Acrobat viewers rebuild the `B` array for all pages of the document.

38. In PDF 1.2, additional-actions dictionaries were inheritable; beginning with PDF 1.3, they are not inheritable.

3.7.1 Content Streams

39. Acrobat viewers report an error the first time they find an unknown operator or an operator with too few operands, but continue processing the content stream. No further errors are reported.

3.9.1 Type 0 (Sampled) Functions

40. When printing, Acrobat performs only linear interpolation, regardless of the value of the `Order` entry.

3.9.2 Type 2 (Exponential Interpolation) Functions

41. Since Type 2 functions are not defined in PDF 1.2 or earlier versions, Acrobat 3.0 (whose native file format is PDF 1.2) reports an Invalid Function Resource error if it encounters a function of this type.

3.9.3 Type 3 (Stitching) Functions

42. Since Type 3 functions are not defined in PDF 1.2 or earlier versions, Acrobat 3.0 (whose native file format is PDF 1.2) reports an error, "Invalid Function Resource," if it encounters a function of this type.

3.9.4 Type 4 (PostScript Calculator) Functions

43. Since Type 4 functions are not defined in PDF 1.2 or earlier versions, Acrobat 3.0 (whose native file format is PDF 1.2) reports an error, "Invalid Function Resource," if it encounters a function of this type.

44. Acrobat uses single-precision floating-point numbers for all real-number operations in a type 4 function.

3.10.2 File Specification Dictionaries

45. .In Acrobat 5.0, file specifications accessed through `EmbeddedFiles` have a `Type` entry whose value is `F` instead of the correct `Filespec`. Acrobat 6.0 and later accept a file specification `whose Type` entry is either `Filespec` or `F`.

46. In certain situations, Acrobat 3.0 and greater do not correctly interpret file specifications referenced by the `F` entry in Form or Image XObjects. Specifically, Acrobat ignores the file specification `EF` entry when that file specification is referenced from the XObject `F` entry.

    The code that follows shows an example of an Image XObject `F` entry referencing a file specification that in turn references an embedded file. Acrobat does not correctly interpret such file specifications.

Example: *Acrobat does not correctly interpret a file specification.*

```
13 0 obj <<
   /Type /XObject
   /Subtype /Image
   /F 12 0 R
```

```
   …
>> stream endstream endobj
12 0 obj <<
   /Type /Filespec
   /EF <</F 10 0 R>>
>> endobj
10 0 obj <<
   /Type /EmbeddedFile
   …
>> stream … endstream endobj
```

4.5.2 Color Space Families

47. If an Acrobat viewer encounters an unknown color space family name, it displays an error specifying the name, but reports no further errors thereafter.

4.5.5 Special Color Spaces (DeviceN Color Spaces)

48. Acrobat viewers support the special meaning of `None` only when a `DeviceN` color space is used as a base color for an indexed color space. For all other uses of `DeviceN`, None is treated as a regular spot color name.

4.5.5 Special Color Spaces (Multitone Examples)

49. This method of representing multitones is used by Adobe Photoshop 5.0.2 and subsequent versions when exporting EPS files. Beginning with version 4.0, Acrobat exports Level 3 EPS files using this method, and can also export Level 1 EPS files that use the "Level 1 separation" conventions of Adobe Technical Note #5044, *Color Separation Conventions for PostScript Language Programs*. These conventions are used to emit multitone images as calls to "customcolorimage" with overprinting, which can then be placed in page layout applications such as Adobe PageMaker®, Adobe In-Design, and QuarkXPress.

4.6 Patterns

50. Acrobat viewers earlier than version 4.0 do not display patterns on the screen, although they do print them to PostScript output devices.

4.7 External Objects

51. Acrobat viewers that encounter an XObject of an unknown type display an error specifying the type of XObject but report no further errors thereafter.

4.8.4 Image Dictionaries

52. Image XObjects in PDF 1.2 and earlier versions are all implicitly unmasked images. A PDF consumer that does not recognize the `Mask` entry treats the image as unmasked without raising an error.

53. All Acrobat viewers ignore the `Name` entry in an image dictionary.

4.8.5 Masked Images

54. Explicit masking and color key masking are features of PostScript LanguageLevel 3. Acrobat 4.0 and later versions do not attempt to emulate the effect of masked images when printing to LanguageLevel 1 or LanguageLevel 2 output devices; they print the base image without the mask.

    The Acrobat 4.0 viewer displays masked images, but only when the amount of data in the mask is below a certain limit. Above that, the viewer displays the base image without the mask.

4.9.1 Form Dictionaries

55. All Acrobat viewers ignore the `Name` entry in a form dictionary.

4.9.3 Reference XObjects

56. Acrobat 8.0 and earlier viewers do not implement reference XObjects. The proxy is always used for viewing and printing.

5.2.5 Text Rendering Mode

57. In Acrobat 4.05 and earlier versions, text-showing operators such as `Tj` first perform the fills for all the glyphs in the string being shown, followed by the strokes for all the glyphs. This produces incorrect results if glyphs overlap.

5.3.2 Text-Showing Operators

58. In versions of Acrobat earlier than 3.0, the horizontal coordinate of the text position after the `TJ` operator paints a character glyph and moves by any specified offset must not be less than it was before the glyph was painted.

59. In Acrobat 4.0 and earlier viewers, position adjustments specified by numbers in a `TJ` array are performed incorrectly if the horizontal scaling parameter, `Th`, is different from its default value of 100.

5.5.1 Type 1 Fonts

60. All Acrobat viewers ignore the `Name` entry in a font dictionary.

61. Acrobat 5.0 and later viewers use the glyph widths stored in the font dictionary to override the widths of glyphs in the font program itself, which improves the consistency of the display and printing of the document. This addresses the situation in which the font program used by the conforming reader is different from the one used by the application that produced the document.

    The font program with the altered glyph widths may or may not be embedded. If it is embedded, its widths should exactly match the widths in the font dictionary. If the font program is not embedded, Acrobat overrides the widths in the font program on the conforming reader's system with the widths specified in the font dictionary.

    It is important that the widths in the font dictionary match the actual glyph widths of the font program that was used to produce the document. Consumers of PDF files depend on these widths in many different contexts, including viewing, printing, fauxing (font substitution), reflow, and word search.

These operations may malfunction if arbitrary adjustments are made to the widths so that they do not represent the glyph widths intended by the PDF producer.

It is recommended that diagnostic and preflight tools check the glyph widths in the font dictionary against those in an embedded font program and flag any inconsistencies. It would also be helpful if the tools could optionally check for consistency with the widths in font programs that are not embedded. This is useful for checking a PDF file immediately after it is produced, when the original font programs are still available.

Note: This implementation note is also referred to in Section 5.6.3, "CIDFonts" (Glyph Metrics in CIDFonts).

5.5.1 Type 1 Fonts (Standard Type 1 Fonts (Standard 14 Fonts))

62. Acrobat 3.0 and earlier viewers may ignore attempts to override the standard fonts.

Names of standard fonts

STANDARD NAMES	ALTERNATIVES
Courier	CourierCourierNew
Courier-Oblique	CourierNew,Italic
Courier-Bold	CourierNew,Bold
Courier-BoldOblique	CourierNew,BoldItalic
Helvetica	Arial
Helvetica-Oblique	Arial,Italic
Helvetica-Bold	Arial,Bold
Helvetica-BoldOblique	Arial,BoldItalic
Times-Roman	TimesNewRoman
Times-Italic	TimesNewRoman
Times-Bold	TimesNewRoman,Bold
Times-BoldItalic	TimesNewRoman,BoldItalic
Symbol	
ZapfDingbats	

Also, Acrobat 4.0 and earlier viewers incorrectly allow substitution fonts, such as TimesNewRoman and ArialMT, to be specified without `FirstChar`, `LastChar`, `Widths`, and `FontDescriptor` entries.

The table titled "Names of standard fonts" shows the complete list of font names that are accepted as the names of standard fonts. The first column shows the proper one (for example, Helvetica); the second column shows the alternative if one exists (for example, Arial).

Acrobat 5 and later viewers do not give special treatment to any font. Any non-embedded font, regardless of name, will be processed in the same way.

5.5.3 Font Subsets

63. For Acrobat 3.0 and earlier viewers, all font subsets whose `BaseFont` names differ only in their tags should have the same font descriptor values and should map character names to glyphs in the same way; otherwise, glyphs may be shown unpredictably. This restriction is eliminated in Acrobat 4.0.

5.5.4 Type 3 Fonts

64. In principle, the value of the `Encoding` entry could also be the name of a predefined encoding or an encoding dictionary whose `BaseEncoding` entry is a predefined encoding. However, Acrobat 4.0 and earlier viewers do not implement this correctly.

65. For compatibility with Acrobat 2.0 and 2.1, the names of resources in a Type 3 font's resource dictionary must match those in the page object's resource dictionary for all pages in which the font is referenced. If backward compatibility is not required, any valid names may be used.

5.6.4 CMaps

66. Embedded CMap files, other than `ToUnicode` CMaps, do not work properly in Acrobat 4.0 viewers; this has been corrected in Acrobat 4.05.

67. Japanese fonts included with Acrobat 6.0 contain only glyphs from the Adobe Japan1-4 character collection. Documents that use fonts containing additional glyphs from the Adobe-Japan1-5 collection must embed those fonts to ensure proper display and printing.

5.7 Font Descriptors

68. Acrobat viewers earlier than version 3.0 ignore the `FontFile3` entry. If a font uses the Adobe standard Latin character set (as defined in Section D.1, "Latin Character Set and Encodings"), Acrobat creates a substitute font. Otherwise, Acrobat displays an error message (once per document) and substitutes any characters in the font with the bullet character.

5.8 Embedded Font Programs

69. For simple fonts, font substitution may be performed using multiple master Type 1 fonts. This substitution can be performed only for fonts that use the Adobe standard Latin character set (as defined in Section D.1, "Latin Character Set and Encodings"). In Acrobat 3.0.1 and later, Type 0 fonts that use a CMap whose `CIDSystemInfo` dictionary defines the Adobe-GB1, Adobe-CNS1 Adobe-Japan1, or Adobe-Korea1 character collection can also be substituted. To make a document portable, fonts that cannot be substituted must be embedded.

6.4.2 Spot Functions

70. When Distiller encounters a call to the PostScript `setscreen` or `sethalftone` operator that includes a spot function, it compares the PostScript code defining the spot function with that of the predefined spot functions shown in Table6.1. If the code matches one of the predefined functions, Distiller puts the name of that function into the halftone dictionary; Acrobat uses that function when printing the PDF document to a PostScript output device. If the code does not match any of the predefined spot

functions, Distiller samples the specified spot function and generates a function for the halftone dictionary; when printing to a PostScript device, Acrobat generates a spot function that interpolates values from that function.

When producing PDF version 1.3 or later, Distiller represents the spot function by using a Type 4 (PostScript calculator) function whenever possible (see Section 3.9.4, "Type 4 (PostScript Calculator) Functions"). In this case, Acrobat uses this function directly when printing the document.

6.5.4 Automatic Stroke Adjustment

71. When drawing to the screen, Acrobat 6.0 always performs automatic stroke adjustment, regardless of the value of the `SA` entry in the graphics state parameter dictionary.

7.5.2 Specifying Blending Color Space and Blend Mode

72. PDF 1.3 or earlier viewers ignore all transparency-related graphics state parameters (blend mode, soft mask, alpha constant, and alpha source). All graphics objects are painted opaquely.

    Note: This implementation note is also referred to in Sections 7.5.3,"Specifying Shape and Opacity" (Mask Shape and Opacity, Constant Shape and Opacity) and 7.5.4, "Specifying Soft Masks" (Soft-Mask Dictionaries).

7.5.3 Specifying Shape and Opacity (Mask Shape and Opacity)

73. PDF 1.3 or earlier viewers ignore the `SMask` entry in an image dictionary. All images are painted opaquely.

    Note: This implementation note is also referred to in Section 7.5.4, "Specifying Soft Masks" (Soft-Mask Images).

8.2.2 Document Outline

74. In PDF 1.2, an additional entry in the outline item dictionary, named `AA`, was defined but was never implemented. Beginning with PDF 1.3, this entry is obsolete and should be ignored.

75. Acrobat viewers report an error when a user activates an outline item whose destination is of an unknown type.

8.3.1 Page Labels

76. Acrobat viewers up to version 3.0 ignore the `PageLabels` entry and label pages with decimal numbers starting at 1.

8.4 Annotations

77. The text of Note 77, as published in the sixth edition of the PDF Reference, is replaced by the following paragraphs.

Interaction between accessibility preference and annotation tab order

This note clarifies the viewer-specific behavior of different Acrobat versions relative to annotation tab order and the accessibility preference selection.

Beginning in PDF 1.5, the user experience of tabbing between annotations on a page (tab order) can be made explicit by means of the page object `Tabs` entry (see Table 3.27). However, when this entry is not defined, Acrobat tab order varies depending on the Acrobat version and on accessibility preference settings. This behavior is viewer-specific because it is not specified by the PDF specification. Because the row, column, and structured tab order settings behave exactly as specified in the PDF specification, they do not have a viewer-specific behavior.

The Acrobat Preferences dialog box presents an Accessibility window. The Tab Order panel in that window has a check box titled "Use document structure for tab order when no explicit tab order is specified." Selecting that check box specifies accessibility preferences.

Regardless of the Acrobat accessibility preference setting, the following applies to (a) all Acrobat versions when processing a PDF document that omits a `Tabs` entry or (b) Acrobat 5 or earlier when processing a PDF document regardless of a `Tabs` entry:

- If the document is structured (contains tagging), the annotations are visited in structure order.
- Otherwise, widgets are visited in the order in which they appear in the `Annots` array, and then other annotation types are visited in row order.

Note: Documents that conform to PDF version 1.5 or earlier have no `Tabs` entries.

Accessibility preference selected (default setting)

The following bulleted items describe how the accessibility preference influences annotation tab order when the accessibility preference is selected:

- If the document is viewed with Acrobat 6 or later and is structured, the `Tabs` entry affects tab order as follows:

  Widget order: When `Tabs` is set to `W`, widgets are visited in the order in which they appear in the `Annots` array and then other annotation types are visited in structure order. Acrobat 9 and later support Widget order.

  All other tab orders: Annotations are visited as specified in the PDF specification supported by the particular version of Acrobat. Acrobat 6 and later support increasingly specific levels of tab order.

- If the document omits a `Tabs` entry or when the document is viewed with Acrobat 5 or earlier, the presence of structure (tagging) affects tab order as follows:

  Structured: Annotations are visited in structure order.

  Not structured: Widgets are visited in the order in which they appear in the `Annots` array, and then other annotation types are visited in row order.

Note: Documents that conform to PDF version 1.5 or earlier have no `Tabs` entries.

Accessibility preference not selected

The following bulleted items explains Acrobat behavior relative to annotation tab order when the accessibility preference is not selected.

The document is viewed with Acrobat 9.

Widget order: When `Tabs` is set to `W`, widgets are visited in the order in which they appear in the `Annots` array, and then other annotation types are visited in row order.

All other tab orders (row, column, structure, and annotation): Annotations are visited as specified by the PDF specification.

No `Tabs` entry: Widget annotations are ordered as they appear in the `Annots` array, followed by other annotation types ordered by rows.

- The document is viewed with Acrobat 6, 7, or 8.

  `Tabs` entry provided: Tab order is as specified by the `Tabs` entry. These versions support only the column, row, and structure order settings.

  No `Tabs` entry or the `Tabs` entry value not recognized: Widgets are visited in the order in which they appear in the `Annots` array, and then other annotation types are visited in row order.

- For a document viewed with Acrobat 5 or earlier, widgets are visited in the order in which they appear in the `Annots` array, and then other annotation types are visited in row order.

Reordering of annotations

For documents that have a `Tabs` entry, Acrobat 6.0 and later reorder annotations in the `Annots` array to match the order described in "Interaction between accessibility preference and annotation tab order" on page 117. The tab order for widgets and other annotation types is then preserved when the document is opened by earlier viewers.

8.4.1 Annotation Dictionaries

78. Acrobat viewers update the annotation dictionary's `M` entry only for text annotations.

79. Acrobat 2.0 and 2.1 viewers ignore the annotation dictionary's `AP` and `AS` entries.

80. All versions of Acrobat through 6.0 ignore the `AP` entry when drawing the appearance of link annotations.

81. Acrobat viewers ignore the horizontal and vertical corner radii in the annotation dictionary's `Border` entry; the border is always drawn with square corners.

82. Acrobat viewers support a maximum of ten elements in the dash array of the annotation dictionary's `Border` entry.

8.4.2 Annotation Flags

83. Acrobat viewers earlier than version 3.0 ignore an annotation's Hidden and Print flags. Annotations that should be hidden are shown; annotations that should be printed are not printed. Acrobat 3.0 ignores the Print flag for text and link annotations.

84. Acrobat 5.0 obeys the Locked flag only for widget annotations. In Acrobat 6.0, markup annotations support it as well.

85. In Acrobat 6.0, the ToggleNoView flag is applicable to mouse-over and selection events.

8.4.3 Border Styles

This note is associated with the description of the `S` *key in Table 8.17.*

86. If an Acrobat viewer encounters a border style it does not recognize, the border style defaults to `S` (Solid).

8.4.4 Appearance Streams

87. Acrobat 5.0 treats the annotation appearance as an isolated group, regardless of whether a `Group` entry is present. This behavior is corrected in Acrobat 6.0.

8.4.5 Annotation Types

88. Acrobat viewers display annotations whose types they do not recognize in closed form, with an icon containing a question mark. Such an annotation can be selected, moved, or deleted, but if the user attempts to activate it, an alert appears giving the annotation type and reporting that a required plug-in is unavailable.

8.4.5 Annotation Types (Link Annotations)

89. Acrobat viewers report an error when a user activates a link annotation whose destination is of an unknown type.

90. When a link annotation specifies a value of `P` for the `H` entry (highlighting mode), Acrobat viewers display the link appearance with a beveled border, ignoring any down appearance that is defined (see Section 8.4.4, "Appearance Streams").

8.4.5 Annotation Types (Polygon and Polyline Annotations)

91. The PDF Reference fifth edition (PDF 1.6) erroneously omitted the `IT` entry from the additional entries specific to a polygon or polyline annotation. This entry specifies the intent of the annotation. (see Table 8.29).

8.4.5 Annotation Types (Text Markup Annotations)

92. In Acrobat 4.0 and later versions, the text is oriented with respect to the vertex with the smallest y value (or the left most of those, if there are two such vertices) and the next vertex in a counterclockwise direction, regardless of whether these are the first two points in the `QuadPoints` array.

8.4.5 Annotation Types (Ink Annotations)

93. Acrobat viewers always use straight lines to connect the points along each path.

8.4.5 Annotation Types (File Attachment Annotations)

94. Acrobat 7.0 and earlier viewers only support file attachment annotations whose referenced file is embedded in the PDF document.

95. Acrobat 7.0 does not include a `Desc` entry in file specifications for file attachment annotations and ignores them if they are present.

8.4.5 Annotation Types (Markup Annotations)

96. PDF 1.7 added the ability to apply markup annotations to 3D views. Although Acrobat 7.0 officially supports PDF 1.6, Acrobat 7.0.7 also supports this feature.

8.4.5 Annotation Types (Watermark Annotations)

97. Watermark annotations are handled correctly only when Acrobat n-up printing is selected. Selecting n-up from within the printer driver does not produce correct results.

8.5.2 Trigger Events

98. In PDF 1.2, the additional-actions dictionary could contain entries named `NP` (next page), `PP` (previous page), `FP` (first page), and `LP` (last page). The actions associated with these entries were never implemented; beginning with PDF 1.3, these entries are obsolete and should be ignored.

99. In PDF 1.2, additional-actions dictionaries were inheritable; beginning with PDF 1.3, they are not inheritable.

100. In Acrobat 3.0, the `O` and `C` events in a page object's additional-actions dictionary are ignored if the document is not being displayed in a page-oriented layout mode. Beginning with Acrobat 4.0, the actions associated with these events are executed if the document is in a page-oriented or single-column layout and are ignored if the document is in a multiple-column layout.

8.5.3 Action Types (Launch Actions)

101. The Acrobat viewer for the Windows platform uses the Windows function ShellExecute to launch an application. The `Win` dictionary entries correspond to the parameters of ShellExecute.

8.5.3 Action Types (URI Actions)

102. URI actions are resolved by the Acrobat WebLink plug-in extension.

103. If the appropriate plug-in extension (WebLink) is not present, Acrobat viewers report the following error when a link annotation that uses a URI action is activated: "The plug-in required by this URI action is not available."

8.5.3 Action Types (Sound Actions)

104. In PDF 1.2, the value of the `Sound` entry was allowed to be a file specification. Beginning with PDF 1.3, this is not supported, but the same effect can be achieved by using an external stream.

105. Acrobat viewers mute the sound if the value of `Volume` is negative; otherwise, this entry is ignored.

106. Acrobat 6.0 does not support the `Synchronous` entry.

107. Acrobat 5.0 and earlier viewers do not support the `Mix` entry.

8.5.3 Action Types (Movie Actions)

108. Acrobat viewers earlier than version 3.0 report an error when they encounter an action of type `Movie`.

8.5.3 Action Types (Hide Actions)

109. Acrobat viewers earlier than version 3.0 report the following error when encountering an action of type `Hide`: "The plug-in needed for this `Hide` action is not available."

110. In Acrobat viewers, the change in an annotation's Hidden flag as a result of a hide action is temporary in that the user can subsequently close the document without being prompted to save changes and the effect of the hide action is lost. However, if the user explicitly saves the document before closing, such changes are saved and thus become permanent.

8.5.3 Action Types (Named Actions)

111. Acrobat viewers earlier than version 3.0 report the following error when encountering an action of type `Named`: "The plug-in needed for this `Named` action is not available."

112. Acrobat viewers extend the list of named actions in Table 8.61 to include most of the menu item names available in the viewer.

8.6.1 Interactive Form Dictionary

113. Acrobat viewers may insert additional entries in the `DR` resource dictionary, such as `Encoding`, as a convenience for keeping track of objects being used to construct form fields. Such objects are not actually resources and are not referenced from the appearance stream.

114. In Acrobat, markup annotations can also make use of the resources in the `DR` dictionary.

115. Acrobat 6.0 recognizes only the stream value of the `XFA` entry; Acrobat 6.02 and later recognize both stream and array values.

8.6.2 Field Dictionaries (Field Names)

116. Beginning in Acrobat 3.0, partial field names cannot contain a period.

117. Acrobat 6.0 and later support Unicode encoding of field names. Versions earlier than Acrobat 6.0 do not support Unicode encoding of field names.

8.6.2 Field Dictionaries (Variable Text)

118. In PDF 1.2, an additional entry in the field dictionary, `DR`, was defined but was never implemented. Beginning with PDF 1.5, this entry is obsolete and should be ignored.

119. If the `MK` entry is present in the field's widget annotation dictionary (see Table 8.39), Acrobat viewers regenerate the entire XObject appearance stream. If `MK` is not present, the contents of the stream outside `/Tx BMC...EMC` are preserved.

8.6.2 Field Dictionaries (Rich Text Strings)

120. To select a font specified by attributes in a rich text string, Acrobat 6.0 follows this sequence, choosing the first appropriate font it finds:

    a. A font in the default resource dictionary (specified by the document's `DR` entry; see Table 8.67) whose font descriptor information matches the font specification in the rich text string. "Font Characteristics" on page 892 describes how this matching is done.

b. A matching font installed on the user's system, ignoring generic font families.

c. A font on the user's system that matches the generic font family, if specified.

d. A standard font (see implementation note 62) that most closely matches the other font specification properties and is appropriate for the current input locale.

8.6.3 Field Dictionaries (Button Fields)

121. The behavior of Acrobat has changed in the situation where a check box or radio button field have multiple children that have the same export value. In Acrobat 4.0, such buttons always turned off and on in unison. In Acrobat 5.0, the behavior of radio buttons was changed to mimic HTML so that turning on a radio button always turned off its siblings regardless of export value. In Acrobat 6.0, the RadiosInUnison flag allows the document author to choose between these behaviors.

8.6.3 Field Types (Choice Fields)

122. In Acrobat 3.0, the `Opt` array must be homogenous: its elements must be either all text strings or all arrays.

8.6.4 Form Actions (Submit-Form Actions)

123. In Acrobat viewers, if the response to a submit-form action uses Forms Data Format (FDF), the URL must end in #FDF so that it can be recognized as such by the Acrobat software and handled properly. Conversely, if the response is in any other format, the URL should not end in #FDF.

8.6.4 "Form Actions (Import-Data Actions)

124. Acrobat viewers set the `F` entry to a relative file specification locating the FDF file with respect to the current PDF document file. If the designated FDF file is not found when the import-data action is performed, Acrobat tries to locate the file in a few well-known locations, depending on the host platform. On the Windows platform, for example, Acrobat searches in the directory from which Acrobat was loaded, the current directory, the System directory, the Windows directory, and any directories listed in the PATH environment variable. On Mac OS, Acrobat searches in the Preferences folder and the Acrobat folder.

125. When performing an import-data action, Acrobat viewers import the contents of the FDF file into the current document's interactive form, ignoring the `F` and `ID` entries in the FDF dictionary of the FDF file.

8.6.4 Form Actions (JavaScript Actions)

126. Because JavaScript 1.2 is not Unicode-compatible, `PDFDocEncoding` and the Unicode encoding are translated to a platform-specific encoding before interpretation by the JavaScript engine.

8.6.6 Forms Data Format (FDF Header)

127. Because Acrobat viewers earlier than 5.0 cannot accept any other version number because of a bug, the FDF file header is permanently frozen at version 1.2. All further updates to the version number will be made via the `Version` entry in the FDF catalog dictionary instead.

8.6.6 Forms Data Format (FDF Catalog)

128. The Acrobat implementation of interactive forms displays the value of the `Status` entry, if any, in an alert note when importing an FDF file.

129. The only `Encoding` value supported by Acrobat 4.0 is Shift-JIS. Acrobat 5.0 supports Shift-JIS, UHC, GBK, and BigFive. If any other value is specified, the default, `PDFDocEncoding`, is used.

8.6.6 Forms Data Format (FDF Fields)

130. Of all the possible entries shown in Table 8.96 on page 717, Acrobat 3.0 exports only the `V` entry when generating FDF, and Acrobat 4.0 and later versions export only the `V` and `AP` entries. Acrobat does, however, import FDF files containing fields that have any of the described entries.

131. If the FDF dictionary in an FDF file received as a result of a submit-form action contains an **F** entry specifying a form other than the one currently being displayed, Acrobat fetches the specified form before importing the FDF file.

132. When exporting a form to an FDF file, Acrobat sets the `F` entry in the FDF dictionary to a relative file specification giving the location of the FDF file relative to that of the file from which it was exported.

133. If an FDF file being imported contains fields whose fully qualified names are not in the form, Acrobat discards those fields. This feature can be useful, for example, if an FDF file containing commonly used fields (such as name and address) is used to populate various types of forms, not all of which necessarily include all of the fields available in the FDF file.

134. As shown in Table 8.96 on page 717, the only required entry in the field dictionary is `T`. One possible use for exporting FDF with fields containing `T` entries but no `V` entries is to indicate to a server which fields are desired in the FDF files returned in response. For example, a server accessing a database might use this information to decide whether to transmit all fields in a record or just some selected ones. As noted in implementation note 133, the Acrobat implementation of interactive forms ignores fields in the imported FDF file that do not exist in the form.

135. The Acrobat implementation of forms allows the option of submitting the data in a submit-form action in HTML Form format for the benefit of existing server scripts written to process such forms. Note, however, that any such existing scripts that generate new HTML forms in response need to be modified to generate FDF instead.

136. When scaling a button's appearance to the bounds of an annotation, versions of Acrobat earlier than 6.0 always took into account the line width used to draw the border, even when no border was being drawn. Beginning with Acrobat 6.0, the `FB` entry in the icon fit dictionary (see Table 8.97 on page 719) allows the option of ignoring the line width.

8.6.6 Forms Data Format (FDF Pages)

137. Acrobat renames fields by prepending a page number, a template name, and an ordinal number to the field name. The ordinal number corresponds to the order in which the template is applied to a page, with 0 being the first template specified for the page. For example, if the first template used on the fifth page has the name Template and has the `Rename` flag set to `true`, fields defined in that template are renamed by prepending the character string `P5.Template_0.` to their field names.

138. Adobe Extreme® printing systems require that the `Rename` flag be `true`.

8.7 Digital Signatures

139. Acrobat computes a byte range digest only when the signature dictionary is referenced from a signature field. There is no byte range signature (that is, there is only an object signature) for FDF file signatures and usage rights signatures referenced from the `UR` entry of a permissions dictionary. In Acrobat 7.0, byte range digests are also computed for usage rights signatures referenced from the `UR3` entry of a permissions dictionary.

    Note: *This note is also attached to the description of the value of the `Changes` key in Table 8.102.*

140. Acrobat 6.0 and later do not provide the `Changes` entry.

8.7.1 Transform Methods

141. Acrobat 6.0 and 7.0 always generate `DigestValue` when creating MDP signatures. Acrobat 6.0 requires the presence of `DigestValue` in order to validate MDP signatures. Acrobat 7.0 does not use `DigestValue` but compares the current and signed versions of the document.

142. Acrobat 6.0 requires a usage rights signature dictionary that is referenced from the `UR` entry of the permissions dictionary in order to validate the usage rights. Acrobat 7.0 supports both `UR` and `UR3`; it uses the `UR3` dictionary if both are present. Adobe PDF generating applications that support PDF 1.6 and greater generate `UR` only for backwards compatibility with Adobe Reader 6.0.

143. In Adobe Reader 6.0, any usage right that permits the document to be modified implicitly enables the FullSave right.

    Adobe Reader 7.0 and 8.0 mimic Reader 6.0 behavior if the PDF document contains a `UR` dictionary but omits a `UR3` dictionary. If the PDF document contains a `UR3` dictionary, only rights specified by the Annots entry that permit the document to be modified implicitly enable the FullSave right. For all other rights, FullSave must be explicitly enabled in order to save the document. (Signature rights permit saving as part of the signing process but not otherwise).

    If the `P` entry in the `UR` transform parameters dictionary is `true`, Acrobat 7.0 and greater conforming readers permit only those rights that are enabled by the entries in the dictionary. However, viewers permit saving the document as long as any rights that permit modifying the document are enabled.

144. In Acrobat 6.0 and greater, the `DigestMethod` entry in the signature reference dictionary is required, even though this specification indicates that entry is optional.

145. In Acrobat 6.0 and greater, the `V` entry in the various transform parameters dictionaries are required, even though this specification indicates otherwise. Those dictionaries include the `DocMDP` transform parameters dictionary, the `UR` transform parameters dictionary, and the `FieldMDP` transform parameters dictionary.

8.7.2 Signature Interoperability

146. In versions earlier than Acrobat 6.0, it was a requirement that the signer's signature be the first certificate in the PKCS#7 object. Acrobat 6.0 removed this restriction, but for maximum compatibility with earlier versions, this practice should be followed.

8.9 Document Requirements

147. PDF 1.7 added the ability to specify requirements the PDF consumer application must satisfy before it can process or display a PDF document. Although Acrobat 7.0 officially supports PDF 1.6, Acrobat 7.0.5 also supports this document requirements feature.

9.1 Multimedia

148. The following media formats are recommended for use in authoring cross-platform PDF files intended for consumption by Acrobat 6.0.

Recommended media types

COMMON EXTENSION	COMMON MIME TYPE	DESCRIPTION
.aiff	audio/aiff	Audio Interchange File Format
.au	audio/basic	NeXT/Sun™ Audio Format
.avi	video/avi	AVI (Audio/Video Interleaved)
.mid	audio/midi	MIDI (Musical Instrument Digital Interface)
.mov	video/quicktime	QuickTime
.mp3	audio/x-mp3	MPEG Audio Layer-3
.mp4	audio/mp4	MPEG-4 Audio
.mp4	video/mp4	MPEG-4 Video
.mpeg	video/mpeg	MPEG-2 Video
.smil	application/smil	Synchronized Multimedia Integration Language
.swf	application/x-shockwave-flash	SWF File Format

9.1.3 Media Clip Objects (Media Clip Data)

149. If the `CT` entry is not present, Acrobat requires a `PL` entry to be present that specifies at least one player that can be used.

9.2 Sounds

150. Acrobat supports a maximum of two sound channels.

9.3 Movies

151. Acrobat viewers do not support the value of `Aspect`.

152. Acrobat viewers support only the `DeviceRGB` and `DeviceGray` color spaces for poster image XObjects. For indexed color spaces with a base color space of `DeviceRGB` (see “Indexed Color Spaces”

on page 262"), Acrobat 5.0 viewers incorrectly treat *hival* as the number of colors rather than the number of colors - 1. Acrobat 6.0 can handle this case properly, as well as the correct value of *hival*; for compatibility with 5.0 viewers, it is necessary to specify *hival* as the number of colors.

Also, Acrobat viewers do not support authoring or rendering posters when the value of `Poster` is `true`.

153. Acrobat viewers treat a `FWScale` value of [999 1] as full screen.

154. Acrobat viewers never allow any portion of a floating window to be offscreen.

9.4 Alternate Presentations

155. The PDF language contains no direct method of initiating an alternate presentation-defined slide show. Instead, a slide show is invoked by a JavaScript call that is typically triggered by an interactive form element. (See Section 8.6, "Interactive Forms.") See the *JavaScript for Acrobat API Reference* (see the Bibliography) for information about starting and stopping a slide show using JavaScript.

156. Beginning with Acrobat 8.1, Acrobat no longer provides support for SVG slide shows.

Acrobat 5.1 through 8.0 support SVG slide shows (MIME content type image/svg+xml). Those versions of Acrobat support the Scalable Vector Graphics (SVG) 1.0 Specification defined by the W3C (see the Bibliography). Implementation notes on support of SVG by Adobe products are available at <http://www.adobe.com/svg/>.

All resources must be either image XObjects (see Section 4.8.4, "Image Dictionaries") or embedded file streams (see Section 3.10.3, "Embedded File Streams").

- Image XObjects used for slide shows must use the DCTDecode filter and a Device RGB color space. Color profile information must be specified in the image XObject dictionary as well as embedded within the stream.
- Embedded audio files must be of type .wav (supported on Windows only, MIME type audio/x-wav) or .mp3 (MIME type audio/mpeg, documented at <http://www.chiariglione.org/mpeg/index.htm>).
- Embedded video must be QuickTime-compatible files of type .avi (MIME type video/ms-video) or .mov (MIME type video/quicktime, documented on Apple's Developer Connection site at <http://developer.apple.com>. To play video, a QuickTime player (version 3 or later) must be installed.

9.5 3D Artwork

157. Acrobat viewers earlier than version 7.0.7 did not honor the `U3DPath` entry of a 3D view dictionary. Acrobat 7.0.7 supports text string values for this entry and deprecates the use of an array for its value.

158. PDF 1.7 introduced several new dictionaries for controlling the appearance and behavior of 3D artwork and for enabling markup annotations on 3D views. Although Acrobat 7.0 officially supports PDF 1.6, Acrobat 7.0.7 also supports these new dictionaries.

10.1 Procedure Sets

159. Acrobat viewers earlier than version 5.0 respond to requests for unknown procedure sets by warning the user that a required procedure set is unavailable and canceling the printing operation. Acrobat 5.0 and later ignores procedure sets.

10.2 Metadata

160. Acrobat viewers display the document's metadata in the Document Properties dialog box and impose a limit of 255 bytes on any string representing one of those values.

10.2.2 Metadata Streams

161. For backward compatibility, applications that create PDF 1.4 documents should include the metadata for a document in the document information dictionary as well as in the document's metadata stream. Applications that support PDF 1.4 or later should check for the existence of a metadata stream and synchronize the information in it with that in the document information dictionary. The Adobe metadata framework provides a date stamp for metadata expressed in the framework. If this date stamp is equal to or later than the document modification date recorded in the document information dictionary, the metadata stream shall be taken as authoritative. If, however, the document modification date recorded in the document information dictionary is later than the metadata stream's date stamp, the document has likely been saved by an application that is not aware of PDF 1.4 metadata streams. In this case, information stored in the document information dictionary should be taken to override any semantically equivalent items in the metadata stream.

10.3 File Identifiers

162. Although the ID entry is not required in a non-encrypted PDF, all Adobe applications that produce PDF files include this entry. Acrobat adds this entry when saving a file if it is not already present.

163. Adobe applications may pass the suggested information to the MD5 message digest algorithm to calculate file identifiers. Note that the calculation of the file identifier need not be reproducible; all that matters is that the identifier is likely to be unique. For example, two implementations of this algorithm might use different formats for the current time, causing them to produce different file identifiers for the same file created at the same time, but the uniqueness of the identifier is not affected.

10.9.2 Content Database (Digital Identifiers)

164. The Acrobat Web Capture plug-in treats external streams referenced within a PDF file as auxiliary data. Such streams are not used in generating the digital identifier.

10.9.3 Content Sets (Image Sets)

165. In Acrobat 4.0 and later versions, if the indirect reference to an image XObject is not removed from the O array when its reference count reaches 0, the XObject is never garbage-collected during a save operation. The image set's reference to the XObject may thus be considered a weak one that is relevant only for caching purposes; when the last strong reference goes away, so does the weak one.

10.9.4 Source Information (URL Alias Dictionaries)

166. Acrobat viewers use an indirect object reference to a shared string for each URL in a URL alias dictionary. These strings can be shared among the chains and with other data structures. It is recommended that other PDF conforming readers adopt this same implementation.

10.10.1 Page Boundaries

167. Acrobat provides various user-specified options for determining how the region specified by the crop box is to be imposed on the output medium during printing. Although these options have varied from one Acrobat version to another, the default behavior is as follows:

    1. Select the media size and orientation according to the operating system's Print Setup dialog. (Acrobat has no direct control over this process.)
    2. Compute an effective crop box by clipping it with the media box and rotating the page according to the page object's `Rotate` entry, if specified.
    3. Center the crop box on the medium, rotating it if necessary to enable it to fit in both dimensions.
    4. Optionally, scale the page up or down so that the crop box coincides with the edges of the medium in the more restrictive dimension.

    The description above applies only in simple printing workflows that lack any other information about how PDF pages are to be imposed on the output medium. In some workflows, there is additional information, either in the PDF file (`BleedBox`, `TrimBox`, or `ArtBox`) or in a separate job ticket (such as JDF or PJTF). In these circumstances, other rules apply, which depend on the details of the workflow.

    Consequently, it is recommended that PDF files initially be created with the crop box the same as the media box (or equivalently, with the crop box omitted). This ensures that if the page is printed on that size medium, the crop box coincides with the edges of the medium, producing predictable and dependable positioning of the page contents. On the other hand, if the page is printed on a different size medium, the page may be repositioned or scaled in implementation-defined or user-specified ways.

10.10.4 Output Intents

168. Acrobat viewers do not make use of the "to CIE" (AToB) information in an output intent's ICC profile.

169. Acrobat 5.0 does not make direct use of the destination profile in the output intent dictionary, but third-party plug-in extensions might do so. Acrobat 6.0 does make use of this profile.

10.10.5 Trapping Support (Trap Network Annotations)

170. Older viewers may fail to maintain the trap network annotation's required position at the end of the Annots array.

171. Older viewers may fail to validate trap networks before printing.

172. In Acrobat 4.0, saving a PDF file with the Optimize option selected causes a page's trap networks to be incorrectly invalidated even if the contents of the page has not been changed. This occurs because the new, optimized content stream generated for the page differs from the original content stream still referenced by the trap network annotation's `Version` array. This problem has been corrected in later versions of Acrobat.

10.10.6 Open Prepress Interface (OPI)

173. The Acrobat 3.0 Distiller application converts OPI comments into OPI dictionaries. When the Acrobat 3.0 viewer prints a PDF file to a PostScript file or printer, it converts the OPI dictionary back to OPI comments. However, the OPI information has no effect on the displayed image or form XObject.

174. Acrobat viewer and Distiller applications earlier than version 4.0 do not support OPI 2.0.

175. In Acrobat 3.0, the value of the `F` entry in an OPI dictionary must be a string.

Appendix C Implementation Limits

176. Acrobat viewers earlier than 5.0 use the PostScript `save` and `restore` operators rather than `gsave` and `grestore` to implement `q` and `Q`, and are subject to a nesting limit of 12 levels.

177. In PDF versions earlier than PDF 1.6, the size of the default user space unit is fixed at 1 / 72 inch. In Acrobat viewers earlier than version 4.0, the minimum allowed page size is 72 by 72 units in default user space (1 by 1 inch); the maximum is 3240 by 3240 units (45 by 45 inches). In Acrobat versions 5.0 and later, the minimum allowed page size is 3 by 3 units (approximately 0.04 by 0.04 inch); the maximum is 14,400 by 14,400 units (200 by 200 inches).

    Beginning with PDF 1.6, the size of the default user space unit may be set with the `UserUnit` entry of the page dictionary. Acrobat 7.0 supports a maximum `UserUnit` value of 75,000, which gives a maximum page dimension of 15,000,000 inches (14,400 * 75,000 * 1 / 72). The minimum `UserUnit` value is 1.0 (the default).

F.2.2 Linearization Parameter Dictionary (Part 2)

178. Acrobat requires a white-space character to follow the left bracket ([ ) character that begins the H array.

179. Acrobat does not currently support reading or writing files that have an overflow hint stream.

    Note: This implementation note is also referred to in Section F.2.5, "Hint Streams (Parts 5 and 10)."

180. Acrobat generates a value for the `E` parameter that incorrectly includes an object beyond the end of the first page as if it were part of the first page.

F.2.6 First-Page Section (Part 6)

181. Acrobat always treats page 0 as the first page for linearization, regardless of the value of `OpenAction`.

F.2.8 Shared Objects (Part 8)

182. Acrobat does not generate shared object groups containing more than one object.

F.3.1 Page Offset Hint Table

183. In Acrobat, items 6 and 7 in the header section of the page offset hint table are set to 0. As a result, item 6 of the per-page entry effectively does not exist; its value is taken to be 0. That is, the sequence of bytes constituting the content stream for a page is described as if the content stream were the first object in the page, even though it is not.

184. Acrobat 4.0 and later versions always set item 8 equal to 0. They also set item 9 equal to the value of item 5, and set item 7 of each per-page hint table entry (Table F.4) to be the same as item 2 of the per-page entry. Acrobat ignores all of these entries when reading the file.

F.3.2 Shared Object Hint Table

185. In Acrobat, item 5 in the header section of the shared objects hint table is unused and is always set to 0.

186. MD5 signatures are not implemented in Acrobat; item 2 in a shared object group entry must be 0.

187. Acrobat does not support more than one shared object in a group; item 4 in a shared object group entry should always be 0.

188. In a document consisting of only one page, items 1 and 2 in the shared object hint table are not meaningful; Acrobat writes unpredictable values for these items.

Implementation Notes to Adobe Extensions to ISO 32000

This section contains notes on the implementation by Acrobat of the Adobe extensions to the PDF language beyond that contained in ISO 32000. The implementation notes for Adobe extensions have a suffix of *E-*.

E-1. Acrobat 9.0 stores data required by the welcome page and header in this way in the `Resources` entry of the collections dictionary. The document *Navigator Template Format*, listed in the Bibliography, describes the content and display of a header and welcome page of a portable collection.

E-2. Acrobat 9.0 Pro has two standard navigators and several custom navigators. When an Acrobat user chooses a custom navigator while in authoring mode, Acrobat embeds the navigator in the portable collection. After being embedded and saved, all editions of Acrobat and Adobe Reader, beginning with version 9.0, can display a portable collection using the embedded custom navigator.

The custom navigator that Acrobat Pro embeds into the portable collection is a stand-alone file available on the local hard drive. A navigator file collects into a single file, the resources required by a navigator of a portable collection. Although PDF 1.7 supports two standard collection views (tile and details), a navigator provides a custom view using a SWF file. The navigator SWF file interacts with a host authoring or viewing application through Acrobat ActionScript API. See the *Acrobat ActionScript API Reference* in the Bibliography.

E-3. The navigator file format, a NAV file, is based on UCF. (See *Uniform Container Format* listed in the Bibliography.) A navigator file contains an XML manifest, the navigator SWF file, and other resources required by the navigator, such as images, localized strings, and private data. See the *Acrobat Navigator File Format* in the Bibliography for information about the format for a navigator file. See also implementation note E-2.

E-4. In Acrobat 9.0, the image referenced by the `Icon` entry may be any size. For best results in Acrobat, the image should be 42 x 42 pixels.

E-5. Acrobat 8.1 adds support for both XFA versions 2.5 and 2.6.

E-6. Authoring forms with barcodes has been available since version 6.0.5; however, the barcode plug-in was not available as a product feature until version 7.0. In version 6.0.5 of Acrobat and Adobe Reader, paper form barcodes could only encode PDF417 symbology. Support for Data Matrix and QR Code symbology was added in version 7.

E-7. The Acrobat implementation of barcode form fields places several constraints on some of the entries in the field dictionary:

Field Flags Entry: In the field dictionary of a barcode form field, bits 1, 3, 13, and 23 of the `Ff` entry (field flags) are set. These bit positions have the names ReadOnly, NoExport, Multiline, and DoNotSpellCheck. (See Tables 8.70 and 8.77 of the *PDF Reference*, *sixth edition, PDF 1.7*.)

Kids Entry: In Acrobat 9, there is a limit of 255 on the array of indirect references to the barcode widget annotations that render the barcode.

The Acrobat implementation of barcode widget annotations sets the `BG` entry of the `MK` dictionary to `[1.0]` to obtain a black background for the barcode widget annotation.

E-8. In Acrobat 9.0, the string length of `WKT` is limited to 1024 bytes.

E-9. For EPSG codes that the Acrobat 9.0 implementation does not support, or for custom coordinate systems not associated with EPSG codes, Well Known Text strings can be used to fully specify an object.

E-10. If the text box, as determined by `TB`, is not large enough, Acrobat 9.0 truncates the string (`UT`).

E-11. In the implementation for version 9.0 of the Adobe Reader software, both Flash and 3D content are supported.

E-12. Rich Media annotations in version 9.0 of the Adobe Reader software support only a subset of the Border Style dictionary (/BS), including the line width and color values. One behavior discrepancy is that if the dictionary is not present, the line width value defaults to zero.

E-13. When the value for the key `Transparent` (as described in "TABLE 9.50d Entries in a RichMediaPresentation dictionary" on page 82) is set to `true`, the Rich Media Annotation artwork is composited over already flattened page content using an alpha channel. Rich Media Annotation artwork is always rendered above all page content and does not, therefore, interfere with the transparency model of the page.

E-14. Older viewers—earlier than 9.0—ignore and do not process the additional approval signature FieldMDP Lock state. To increase compatibility, Acrobat sets `FieldMDP` to lock all fields when the document is locked. Consequently, older viewers consider all fields locked when the document lock is applied using Acrobat 9.0. However, adding annotations is still possible.

Documents signed with Acrobat 9.0 with locking, and then opened and modified in an older viewer (adding annotations), and then opened in Acrobat 9.0 show invalidated signatures.

E-15. How seed values are interpreted is both application-specific and specific to the signature handler associated with the signature field. Each application and handler may choose how to honor the seed values. The behavior of Acrobat and the PPKLite signature handler is documented below for the `LockDocument` seed value, and for `AppearanceFilter` in implementation note E-15.

A seed value of `auto` specifies that the implementation is application-specific. Acrobat processes the `auto` value as follows: If the document contains hidden fields, the signer is not allowed to lock the document. Otherwise, the signer can choose whether to lock at the time of signing, with the same behavior as the value being set to `false` and the required flag not set. Acrobat does not honor the required flag when the `LockDocument` value is `auto`.

E-16. The `AppearanceFilter` is implemented as follows: If the `AppearanceFilter` is present, it automatically chooses a matching appearance from the user interface. If the `AppearanceFilter` is optional, and if a matching appearance is not available, it defaults to the normal behavior where any appearance can be chosen. If the filter is required, the matching appearance is selected and the selection box disabled.

E-17. Version 9.0 of Acrobat and Adobe Reader do not implement the value of `A` (annotation array order) of the `Tabs` entry of the page object.

E-18. Acrobat Pro and Acrobat Pro Extended while in portable collection editing mode use the combination of `ID`, `Locale`, and `Version` to uniquely identify a navigator. This behavior allows the application to determine that the navigator embedded in a collection is the same as a navigator description stored on hard disk. IDs can be human-readable or a URN based on a machine-generated GUID, as long as they are likely to be unique. GUIDs are defined in RFC4122, *A Universally Unique Identifier (UUID) URN Namespace*, listed in the Bibliography.

E-19. For version 9.0, a PDF viewing application loads the navigator found in the PDF file, no matter what navigators are available.

In addition to loading the navigator in the PDF file, a portable collection authoring application—Acrobat Pro or Acrobat Pro Extended—loads the navigator with the highest version number when it finds more than one with the same ID and locale. Acrobat, in authoring mode, usually shows the PDF file's navigator as well as all the loaded navigators. If one of the loaded navigators matches the PDF file's navigator (ID, version, locale), only one copy is shown in the navigator list. If the PDF file's navigator is newer (higher version number), only it is shown. If the PDF file's navigator is older, both are shown, with the loaded navigator marked as *new*.

E-20. For Acrobat 9.0, if a `WKT` string is present, it is used to specify the coordinate system. If an `EPSG` code is present, it shall be consistent with the `WKT` string. If no `WKT` string is present, the `EPSG` code is used to specify the coordinate system.

Index

Number

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X