Personal Folder File (PFF) file format specification

Summary

***PFF is short for Personal Folder File and is mainly used by Microsoft** Outlook to store e-mails, appointments, contacts, tasks, etc. This specification is based on the work by libpst* [SMITH02] *started in 2002 and was enhanced by analyzing test data in 2008 and 2009. In 2010 it was synced with Microsoft's official PST specification* [MS-PST].

This document is intended as a working document for the Personal Folder File format specification. Which should allow existing Open Source forensic tooling to be able to process this file type.

Document information

Author(s):	Joachim Metz <joachim.metz@gmail.com>
Abstract:	This document contains information about the Personal Folder File format.
Classification:	Public
Keywords:	PFF, Personal Folder File, OFF, Offline Folder File, PAB, Personal Address Book, PST, Personal Storage Table, OST, Outlook Storage Table

License

Revision history

Version	Author	Date	Comments
0.0.1	J.B. Metz	June 2008	Initial version based on earlier notes.

Version	Author	Date	Comments
0.0.2	J.B. Metz	June 2008	Added information about LZFu compression and arrays.
0.0.3	J.B. Metz	July 2008	Added information about allocation tables.
0.0.4	J.B. Metz	October 2008	Updated for initial release.
0.0.5	J.B. Metz	October 2008	Added reference about RTF compression.
0.0.6	J.B. Metz	October 2008	Addition about attachments of type embedded object.
0.0.7	J.B. Metz	October 2008	Addition about 0x85 0x85 index node type.
0.0.8	J.B. Metz	October 2008	Added information about descriptor list type.
0.0.9	J.B. Metz	November 2008	Added information about name-to-id map.
0.0.10	J.B. Metz	December 2008	Additional information about the header.
0.0.11	J.B. Metz	December 2008	Additional information about the header and item values types (property types).
0.0.12	J.B. Metz	December 2008	Additional information about item types (property names/identifiers) and the name-to-id map.
0.0.13	J.B. Metz	December 2008	Additional information about the file header and the item types.
0.0.14	J.B. Metz	December 2008	Additional information about the item types.
0.0.15	J.B. Metz	January 2009	Additional information about the allocation full maps.
0.0.16	J.B. Metz	January 2009	Additional information about HTML e-mail body type.
0.0.17	J.B. Metz	January 2009	Additional information about name-to-id map.
0.0.18	J.B. Metz	January 2009	Moved MAPI definitions to separate document.
0.0.19	J.B. Metz	March 2009	Added information encountered by C. Byington of the libpst project. Renamed the local descriptor list into the local descriptors. Added information about the build-in Public strings class in the name-to-id map.
0.0.20	J.B. Metz	March 2009	Additional information and corrections.

Version	Author	Date	Comments
0.0.21	J.B. Metz	May 2009	Update for recipient types based on patch by K. Mazur
0.0.22	J.B. Metz	May 2009	Update for attachment rendering position from finding by K. Mazur. Changed local descriptors node level into node type.
0.0.23	J.B. Metz	June 2009	Update for non UTF-16 strings in name-to-id map. Clean up of the PFF items, mainly provided for in the MAPI documentation. First table index offset of an Outlook 2007 SP2 ost file is out of the ordinary.
0.0.24	J.B. Metz	June 2009	Added information about the b5 table header table entries level. Added information about the local descriptors type (level of indirection).
0.0.25	J.B. Metz	June 2009	Added information about 6c and 8c table. Added information about sub folders item.
0.0.26	J.B. Metz	July 2009	Additional information and corrections. Added information about sub messages item. Added missing information about 64-bit array type
0.0.27	J.B. Metz	September 2009	Added information about none encrypted pst files with encrypted data.
0.0.28	J.B. Metz	September 2009	Added information about the array (indirection) level.
0.0.29	J.B. Metz	September 2009	Added information about the bc table (indirection) level.
0.0.30	J.B. Metz	January 2010	Small changes
0.0.31	J.B. Metz	January 2010	Additional information based on [MS-PST].
0.0.32	J.B. Metz	March 2010	Corrected information about table offset index entries.
0.0.33	J.B. Metz	April 2010	Updated remarks.
0.0.34	J.B. Metz	April 2010	Added codepage 1200 scenario.
0.0.35	J.B. Metz	June 2010	Email change
0.0.36	J.B. Metz	July 2010	Changes to local descriptors
0.0.37	J.B. Metz	August 2010	Changed table entry to record entry, and table entry definition to column definition for clarity.
0.0.38	J.B. Metz	January 2010	License version update
0.0.39	J.B. Metz	July 2012	Email change

Version	Author	Date	Comments
0.0.40	J.B. Metz	August 2012	Updated references.
0.0.41	J.B. Metz	February 2013	Small changes.
0.0.42	J.B. Metz	February 2013	Changes for Outlook 2013 OST (64-bit 4k page) file with thanks to S. Gurjar.
0.0.43	J.B. Metz	July 2013	Additional information about 64-bit 4k page format.
0.0.44	J.B. Metz	August 2013	Additional information about corruption scenarios with thanks to J.M. Cabo.
0.0.45	J.B. Metz	August 2013	Additional information about Outlook 2013 OST (64-bit 4k page) format with thanks to I. Rogov.
0.0.46	J.B. Metz	August 2013	Additional information about Outlook 2013 OST (64-bit 4k page) format.
0.0.47	J.B. Metz	July 2018	Switched to asciidoc format.
0.0.48	J.B. Metz	July 2020	Changes for formatting.

1. Overview

The PFF (Personal Folder File) and OFF (Offline Folder File) format is used to store Microsoft Outlook e-mails, appointments and contacts. The OST (Offline Storage Table), PAB (Personal Address Book) and PST (Personal Storage Table) file format consist of the PFF format. A PFF consist of the following distinguishable elements:

- file header
- file header data
- index branch node
- index leaf node
- (file) offset index
- (item) descriptor index
- local descriptors
- item table type

Characteristics	Description
Byte order	little-endian
Date and time values	FILETIME in UTC

Characteristics	Description
Character strings	ASCII strings are Single Byte Character (SBC) or Multi Byte Character (MBC) string stored with a codepage. Sometimes referred to as ANSI string representation. Though technically maybe incorrect, this document will use term (extended) ASCII string. Unicode strings are stored in UTF-16 little-endian without the byte order mark (BOM).

Certain elements of the PFF format are related to the Microsoft (Office) Outlook Messaging API (MAPI).

[MS-PST] defines two types of the PFF:

- the 32-bit ANSI format
- the 64-bit Unicode format

A third variant was discovered in an Outlook 2013 OST file namely:

- the 64-bit Unicode format with 4k (4096 bytes) pages.

1.1. Test version

Files created by the following version of programs were used to test the information within this document:

- Microsoft Outlook 2000
- Microsoft Outlook 2003
- Microsoft Outlook 2007
- Microsoft Outlook 2010
- Microsoft Outlook 2013
- Exmerge
- Scanpst

2. File header

The file header common to both the 32-bit and 64-bit PFF format consists of 24 bytes and consists of:

Offset	Size	Value	Description
0	4	"\x21\x42\x44\x4e" (!BDN)	The signature (magic identifier)

Offset	Size	Value	Description
4	4		A weak CRC32 of the following 471 bytes In 64-bit files this CRC seems to be ignored because of the CRC at the end of the file header data at offset 524.
8	2		The content type (client signature) See section: Content types
10	2		The data version (NDB version) NDB is short for node database See section: Format types
12	2		Content version (Client version) **Unknown use**
14	1	0x01	Creation Platform **Unknown use** must be 0x01 according to [MS-PST] Seen 0x02 found in scanpst recovered pst
15	1	0x01	Access Platform **Unknown use** must be 0x01 according to [MS-PST] Seen 0x02 found in scanpst recovered pst
16	4	0	**Unknown (dwOpenDBID)** Reserved, sometimes contains: 0x40 0x00 0x00 0x00 (unclean unmount?)
20	4	0	**Unknown (dwOpenClaimID)** Reserved, (mostly empty) (unclean unmount?)

2.1. Content types

Value	Identifier	Description
"\x41\x42" (AB)		Used for PAB files
"\x53\x4d" (SM)		Used for PST files
"\x53\x4f" (SO)		Used for OST files

2.2. Format types

Value	Identifier	Description
14		32-bit ANSI format
15		32-bit ANSI format

Value	Identifier	Description
21		64-bit Unicode format (by Visual Recovery)
23		64-bit Unicode format
36		64-bit Unicode format with 4k

2.3. The 32-bit header data

The 32-bit header data is 488 bytes of size and consists of:

Offset	Size	Value	Description
24	4		Next (available) index pointer
28	4		Next (available) index back pointer **In more recent pst/ost files used for the density list at offset 0x4200**
32	4		Seed value Unique value for the CRC calculation, which changes for consecutive created files
36	128 (32 x 4)		Descriptor index high water marks (NID high-water marks)
Part of the header data to which [MS-PST] refers to as the root			
164	4	0	**Unknown (Reserved)**
168	4		Total file size
172	4		Last data allocation table offset The file offset to the last data allocation table
176	4		Total available data size
180	4		Total available page size
184	4		The descriptor index back pointer the value that should appear in the parent offset of the root node of the descriptor index b-tree
188	4		The descriptor index file offset File offset of the the of the descriptor index b-tree
192	4		The (file) offset index back pointer the value that should appear in the parent offset of the root node of the (file) offset index b-tree
196	4		The (file) offset index file offset File offset of the the of the (file) offset index b-tree
200	1		Allocation table validation type See section: Allocation table validation types
201	1	0	**Unknown (Reserved)**
202	2	0	**Unknown (Reserved)**

Offset	Size	Value	Description
End of the root			
204	128		The initial data free map
332	128		The initial page free map
460	1	0x80	Senitinal
461	1		Encryption type See section: Encryption types
462	2	0	**Unknown (Reserved)** **In older formats (rgbReserved Index) which is 17 bytes of size**
464	8	0	**Unknown (Reserved)**
472	4	0	**Unknown (Reserved)**
476	3	0	**Unknown (Reserved)**
479	1	0	**Unknown (Reserved)**
480	32	0	**Unknown (Reserved)**

Data after file header data probably extended data for AMap

Offset	Size	Value	Description
512	4		**Unknown value** **Changes consecutive created pst files**
516	4		**Unknown value** **Changes consecutive created pst files**
520	4		**Unknown value** **Does not change in consecutive created pst files**
524	4		**Unknown value** **Changes consecutive created pst files**
528	16880		**Empty values**

2.4. The 64-bit header data

The 64-bit header data is 540 bytes of size and consists of:

Offset	Size	Value	Description
24	8		**Unused (bidUnused)** Sometimes contains: 0x04 0x00 0x00 0x00 0x01 0x00 0x00 0x00
32	8		Next (available) index back pointer In more recent pst/ost files used for the density list at offset 0x4200

Offset	Size	Value	Description
40	4		Seed value Unique value for the CRC calculation, which changes for consecutive created files
44	128 (32 x 4)		Descriptor index high water marks (NID high-water marks)
172	8	0	**Unknown (qwAlign)** **Unused**
Part of the header data to which [MS-PST] refers to as the root			
180	4	0	**Unknown (cOrphans)** **Reserved**
184	8		Total file size
192	8		Last data allocation table offset The file offset to the last data allocation table
200	8		Total available data size
208	8		Total available page size
216	8		The descriptor index back pointer The value that should appear in the parent offset of the root node of the descriptor index b-tree
224	8		The descriptor index file offset File offset of the the of the descriptor index b-tree
232	8		The (file) offset index back pointer The value that should appear in the parent offset of the root node of the (file) offset index b-tree
240	8		The (file) offset index file offset File offset of the the of the (file) offset index b-tree
248	1		Allocation table validation type See section: Allocation table validation types
249	1	0	**Unknown (bARVec)** **Reserved**
250	2	0	**Unknown (cARVec)** **Reserved**
End of the root			
252	4	0	**Unknown (dwAlign)** Alignment data according to [MS-PST]
256	128	0xff	The initial data free map According to [MS-PST] deprecated
384	128	0xff	The initial free page map According to [MS-PST] deprecated
512	1	0x80	Senitinal

Offset	Size	Value	Description
513	1		Encryption type See section: Encryption types
514	2	0	**Unknown (bReserved)** **Reserved**
516	8		Next (available) index pointer
524	4		A weak CRC32 of the previous 516 bytes
528	3	0	**Unknown (rgbVersionEncoded)** **Reserved**
531	1	0	**Unknown (bLockSemaphore)** **Reserved**
532	32	0	**Unknown (rgbLock)** **Reserved**

Data after file header data probably extended data for AMap

Offset	Size	Value	Description
538	8		**Unknown value**
546	8		**Unknown value**
552	...		

2.5. Allocation table validation types

Value	Identifier	Description
0x00	INVALID_AMAP	One or more allocation tables are invalid
0x01	VALID_AMAP1	All allocation tables are valid According to [MS-PST] this value is deprecated
0x02	VALID_AMAP2	All allocation tables are valid

2.6. Descriptor index high water marks

```
Unknown use
NID => Node ID ? Seems to be the equivalent of the items identifiers
type=n is the number in the array

Under high-water mark, any object less than the user's security level can be
opened, but the object is relabeled to reflect the highest security level
currently open. Hence the name.
```

2.6.1. Descriptor index high water mark type

Value	Identifier	Description
0x00000400	NID_TYPE_NORMAL_FOLDER	Folder or any other type
0x00004000	NID_TYPE_SEARCH_FOLDER	Search folder
0x00008000	NID_TYPE_ASSOC_MESSAGE	Associated content
0x00010000	NID_TYPE_NORMAL_MESSAGE	Message

2.7. Encryption types

Value	Identifier	Description
0x00	NDB_CRYPT_NONE	No encryption
0x01	NDB_CRYPT_PERMUTE	Compressible encryption According to [MS-PST] this is encryption with 'permutation algorithm', which is a substitution cipher
0x02	NDB_CRYPT_CYCLIC	High encryption According to [MS-PST] this is encryption with 'cyclic algorithm', which is similar to the 3 rotor Enigma cipher

3. Pages

[MS-PST] defines a common structure for the allocation table, the index b-tree, the free map and the density list as the page.

3.1. The 32-bit page

The 32-bit page is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	500		Page data
500	1		Page type See section: Page types
501	1		Copy of page type

Offset	Size	Value	Description
502	2		Signature
504	4		The back pointer
508	4		A weak CRC32 of the 496 bytes of the table data

3.2. The 64-bit page

The 64-bit page is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	496		Page data
496	1		Page type See section: Page types
497	1		Copy of page type
498	2		Signature
500	4		A weak CRC32 of the 496 bytes of the table data
504	8		The back pointer

3.3. The 64-bit 4k page

The 64-bit 4k page is 4096 bytes of size and consists of:

Offset	Size	Value	Description
0	496		Page data
4072	1		Page type See section: Page types
4073	1		Copy of page type
4074	2		Signature
4076	4		A weak CRC32 of the 4072 bytes of the table data
4080	8		The back pointer
4088	8		**Unknown**

3.4. Page types

Value	Identifier	Description
0x80	ptypeBBT	Offset index b-tree node
0x81	ptypeNBT	Descriptor index b-tree node
0x82	ptypeFMap	Free map

Value	Identifier	Description
0x83	ptypePMap	Page allocation table
0x84	ptypeAMap	Data allocation table
0x85	ptypeFPMap	Free page map
0x86	ptypeDL	Density list

4. The allocation table

The PFF contains several allocation tables. These tables are used to describe what parts of the PFF are in use and free.

4.1. The 32-bit allocation table

The 32-bit allocation is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	4		**Unknown (Padding)**
4	496		The allocation table data Each bit represents a certain number of bytes (block). A value of 1 means that the block is allocated, 0 if not
Footer 12 bytes of size			
500	1	0x83 0x84	Page type See section: Page types and Allocation table types
501	1	0x83 0x84	Copy of page type
502	2	0	Signature According to [MS-PST] this should be empty
504	4		The back pointer The value contains the allocation table offset
508	4		A weak CRC32 of the 496 bytes of the allocation table data

4.2. The 64-bit allocation table

The 64-bit allocation is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	496		The allocation table data Each bit represents a certain number of bytes (block). A value of 1 means that the block is allocated, 0 if not
Footer 16 bytes of size			

Offset	Size	Value	Description
496	1	0x83 0x84	Page type See section: Page types and Allocation table types
497	1	0x83 0x84	Copy of page type
498	2		Signature According to [MS-PST] this should be empty
500	4		A weak CRC32 of the 496 bytes of the allocation table data
504	8		The back pointer The value contains the allocation table offset

4.3. The 64-bit 4k page allocation table

Offset	Size	Value	Description
0	4072		The allocation table data Each bit represents a certain number of bytes (block). A value of 1 means that the block is allocated, 0 if not
Footer 24 bytes of size			
4072	1	**0x83** 0x84	Page type See section: Page types and Allocation table types
4073	1	**0x83** 0x84	Copy of page type
4074	2		Signature
4076	4		A weak CRC32 of the first 4072 bytes of the allocation table data
4080	8		Back pointer The value contains the allocation table offset
4088	8		**Unknown**

4.4. Allocation table types

For both the 32-bit ANSI format and the 64-bit Unicode format the behavior of the allocation tables is as following:

- The allocation table at offset 0x4400 with page type 0x84 addresses 64 byte blocks. Where the first bit in the allocation table data refers to offset 0x4400. These are used for the data allocation. The tables repeat themselves every 496 x 8 x 64 = 253952 bytes.
- The allocation table at offset 0x4600 with page type 0x83 addresses 512 byte blocks. Where the first bit in the allocation table data refers to offset 0x4400. These are used for the page allocation. The tables repeat themselves every 496 x 8 x 512 = 2031616 bytes.

For the 64-bit Unicode format with 4k (4096 bytes) pages format the allocation tables is as following:

- The allocation table at offset 0x22000 with page type 0x84 addresses 64 byte blocks. Where the first bit in the allocation table data refers to offset 0x22000. These are used for the data allocation. The tables repeat themselves every 4072 x 8 x 512 = 16678912 bytes.

NOTE	Page type 0x83 not yet been seen to be used with the 64-bit Unicode format with 4k format.

5. The index b-tree

The PFF consists of multiple index b-trees.

- The (file) offset index b-tree (Block B-Tree (BBT))
- The (item) descriptor index b-tree (Node B-Tree (NBT))

These b-trees have a similar basic structure.

An index b-tree consists of:

- branch nodes that point to branch or leaf nodes
- leaf nodes that contain the index data

5.1. The 32-bit index b-tree node

Both the 32-bit branch and leaf node have a similar structure which is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	496		Node entries (number of records x entry size) Maximum of 496 the remaining values are zeroed
Footer 16 bytes of size			
496	1		The number of entries The number of entries that are used
497	1		The maximum number of entries
498	1		The size of an entry
499	1		Node level A zero value represents a leaf node A value greater than zero branch nodes with the highest level representing the root
500	1	0x80 0x81	Page type See section: Page types

Offset	Size	Value	Description
501	1	0x80 0x81	Copy of page type
502	2		Signature
504	4		Back pointer must match the back pointer that pointed to this node
508	4		A weak CRC32 of the first 500 bytes of the index node

5.1.1. The 32-bit index b-tree branch node entry

The 32-bit index b-tree node entry is used in branch nodes. It is 12 bytes of size and consists of:

Offset	Size	Value	Description
0	4		The index identifier of the first child node Identifier of type node identifier See section: Index identifier
4	4		The back pointer
8	4		The (file) offset

The index b-tree node will contain the following values:

- The maximum number of entries: 41
- The size of an entry: 12

An index b-tree node can contain the same identifier value as a (file) offset index entry. This occurs when the leaf node is the lowest identifier in the branch node.

5.1.2. The 32-bit (file) offset index entry

The 32-bit (file) offset index entry is used in leaf nodes. It is 12 bytes of size and consists of:

Offset	Size	Value	Description
0	4		The identifier Identifier of type block identifier
4	4		The (file) offset
8	2		The size
10	2		The reference count

The index b-tree node will contain the following values:

- The maximum number of entries: 41
- The size of an entry: 12

The first LSB of the identifier is reserved.

The second LSB of the identifier is used to indicate if the block is internal or not.

- 0 = is not internal (external)
- 1 = is internal (used for array and local descriptors)

In an encrypted PFF the internal flag also indicates if the corresponding entry is encrypted or not. See section: Block types for more information.

When the index tree is searched make sure to clear the first LSB in the identifier.

5.1.3. The 32-bit descriptor index b-tree leaf node entry

The 32-bit descriptor index b-tree leaf node entry is 16 bytes of size and consists of:

Offset	Size	Value	Description
0	4		The (descriptor) index identifier Identifier of type node identifier See section: Index identifier
4	4		The (file) offset index identifier of the data
8	4		The (file) offset index identifier of the local descriptors
12	4		The parent (descriptor) index identifier

The index b-tree node will contain the following values:

- The maximum number of entries: 31
- The size of an entry: 16

5.2. The 64-bit index b-tree node

Both the 64-bit branch and leaf node have a similar structure which is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	488		Node entries (number of records x entry size) Maximum of 488 the remaining values are zeroed
Footer 24 bytes of size			
488	1		The number of entries The number of entries that are used
489	1		The maximum number of entries
490	1		The size of an entry

Offset	Size	Value	Description
491	1		Node level A zero value represents a leaf node A value greater than zero branch nodes with the highest level representing the root
492	4		**Unknown (Padding)**
496	1	0x80 0x81	Page type See section: Page types
497	1	0x80 0x81	Copy of page type
498	2		Signature
500	4		A weak CRC32 of the first 496 bytes of the index node
504	8		Back pointer must match the back pointer that pointed to this node

5.2.1. The 64-bit index b-tree branch node entry

The 64-bit index b-tree node entry is used in branch nodes. It is 24 bytes of size and consists of:

Offset	Size	Value	Description
0	8		The index identifier of the first child node Identifier of type node identifier, only 32-bit are used See section: Index identifier
8	8		The back pointer
16	8		The (file) offset

The index b-tree node will contain the following values:

- The maximum number of entries: 20
- The size of an entry: 24

An index b-tree node can contain the same identifier value as a (file) offset index entry. This occurs when the leaf node is the lowest identifier in the branch node.

5.2.2. The 64-bit (file) offset index entry

The 64-bit (file) offset index entry is used in leaf nodes. It is 24 bytes of size and consists of:

Offset	Size	Value	Description
0	8		The index identifier Identifier of type block identifier
8	8		The (file) offset
16	2		The size

Offset	Size	Value	Description
18	2		The reference count
20	4		File offset of the data allocation table

The index b-tree node will contain the following values:

- The maximum number of entries: 20
- The size of an entry: 24

The first LSB of the identifier is reserved.

The second LSB of the identifier is used to indicate if the block is internal or not.

- 0 = is not internal (external)
- 1 = is internal (used for array and local descriptors)

In an encrypted PFF the internal flag also indicates if the corresponding entry is encrypted or not. See section: 7.4 Block type for more information.

When the index tree is searched make sure to clear the first LSB in the identifier.

5.2.3. The 64-bit descriptor index b-tree leaf node entry

The 64-bit descriptor index b-tree leaf node entry is 32 bytes of size and consists of:

Offset	Size	Value	Description
0	8		The (descriptor) index identifier Identifier of type node identifier, only 32-bit are used See section: Index identifier
8	8		The (file) offset index identifier of the data
16	8		The (file) offset index identifier of the local descriptors
24	4		The parent (descriptor) index identifier
28	4		**Unknown** **This value mainly contains 2, unless when both the data and local descriptor are empty.**

The index b-tree node will contain the following values:

- The maximum number of entries: 15
- The size of an entry: 32

5.3. The 64-bit 4k page index b-tree node

In Outlook 2013, at least for OST files, a 4k (4096 bytes) page version of the 64-bit index b-tree node was introduced.

Both the 64-bit branch and leaf node have a similar structure which is 4096 bytes of size and consists of:

Offset	Size	Value	Description
0	4056		Node entries (number of records x entry size) Maximum of 488 the remaining values are zeroed
Footer 40 bytes of size			
4056	2		The number of entries The number of entries that are used
4058	2		The maximum number of entries
4060	1		The size of an entry
4061	1		Node level A zero value represents a leaf node A value greater than zero branch nodes with the highest level representing the root
4062	10		**Unknown (Padding)**
4072	1	0x80 0x81	Page type See section: Page types
4073	1	0x80 0x81	Copy of page type
4074	2		Signature
4076	4		A weak CRC32 of the first 4072 bytes of the index node
4080	8		Back pointer must match the back pointer that pointed to this node
4088	8		**Unknown**

The node entry structures are the same as those of the 64-bit index b-tree node (512 byte page) version.

5.4. Index identifier

The index identifier is 32-bit of size and consists of:

Offset	Size	Value	Description
0.0	5 bits		Identifier type See section: Node identifier types
0.5	27 bits		Identifier value

NOTE The identifiers should be unique and are so for allocated descriptors. However unallocated descriptors can have identifiers that are in use.

5.4.1. Node identifier types

The node identifier is used in both the item descriptor identifier and the table value reference. It signifies the type of node the identifier is referencing.

Value	Identifier	Description
0x00	NID_TYPE_HID	Table value (or heap node) See section: The table
0x01	NID_TYPE_INTERNAL	Internal node See section: Internal nodes
0x02	NID_TYPE_NORMAL_FOLDER	Folder item
0x03	NID_TYPE_SEARCH_FOLDER	Search folder item
0x04	NID_TYPE_NORMAL_MESSAGE	Message item
0x05	NID_TYPE_ATTACHMENT	Attachment item
0x06	NID_TYPE_SEARCH_UPDATE_QUEUE	Queue of changed search folder items
0x07	NID_TYPE_SEARCH_CRITERIA_OBJECT	Search folder criteria
0x08	NID_TYPE_ASSOC_MESSAGE	Associated contents item
0x0a	NID_TYPE_CONTENTS_TABLE_INDEX	**Unknown** **Internal, Persisted View- related**
0x0b	NID_TYPE_RECEIVE_FOLDER_TABLE	Inbox item (or received folder table)
0x0c	NID_TYPE_OUTGOING_QUEUE_TABLE	Outbox item (or outgoing queue table)

Value	Identifier	Description
0x0d	NID_TYPE_HIERARCHY_TABLE	Sub folders item (or hierarchy table) See section: The related sub folders item
0x0e	NID_TYPE_CONTENTS_TABLE	Sub messages item (or contents table) See section: The related sub messsages item
0x0f	NID_TYPE_ASSOC_CONTENTS_TABLE	Sub associated contents item (or associated contents table) See section: The related sub associated contents item
0x10	NID_TYPE_SEARCH_CONTENTS_TABLE	Search contents table **Consists of an ac table**
0x11	NID_TYPE_ATTACHMENT_TABLE	Attachments item Consists of a 7c table
0x12	NID_TYPE_RECIPIENT_TABLE	Recipients item Consists of a 7c table
0x13	NID_TYPE_SEARCH_TABLE_INDEX	**Unknown** **Internal, Persisted View- related**
0x14		**Unknown** **Related + 18 folder item** **Consists of a 8c table**
0x15		**Unknown** **Related + 19 folder item** **Consists of a 8c table**
0x16		**Unknown** **Unknown 1718 sub item** **Consists of a 7c table**
0x17		**Unknown** **Unknown 1751 sub item**
0x18		**Unknown** **Unknown 1784 sub item**
0x1f	NID_TYPE_LTP	Local descriptor value See section: The local descriptors

6. The free map

The free map contains information about the longest consecutive number of bytes in the data allocation tables.

According to [MS-PST] the free maps should not be used. The density list should be used instead.

6.1. The 32-bit free map

NOTE	According to [MS-PST] the page free map only has a 64-bit format. However 32-bit PFF have been seen containing page type 0x85. See notes below.

The 32-bit free map is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	4		**Unknown (Padding)**
4	496		The free map data
500	1	0x82 0x85	Page type See section: Page types
501	1	0x82 0x85	Copy of page type
502	2		Signature According to [MS-PST] this should be empty
504	4		The back pointer The value is the free map offset
508	4		A weak CRC32 of the 496 bytes of the free map data

6.2. The 64-bit free map

The 64-bit free map is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	496		The free map data
496	1	0x82	0x85
Page type See section: Page types	497	1	0x82
0x85	Copy of page type	498	2

Offset	Size	Value	Description
	Signature According to [MS-PST] this should be empty	500	4
	A weak CRC32 of the 496 bytes of the free map data	504	8

6.3. The 64-bit 4k page free map

TODO, not seen so far

6.4. Free map types

The free map with page type 0x82 addresses the maximum number of continuous free data blocks in the corresponding data allocation table. Every byte in the free map data represents a separate data allocation table.

The free map with page type 0x85 addresses free pages. Every bit in the free map data represents a separate page allocation table.

6.5. Notes

Page type 0x85 seen in 32-bit PFF.

Offset	Size	Value	Description
0	4		Next node back pointer must match the back pointer of the next node
4	4		Next node offset
8	488		**Unknown values** Maximum of 488 the remaining values are zeroed
496	1		**Unknown value** **Seen: 0x00 in most nodes, 0x40**
497	1		**Unknown value** **Seen: 0x00 in most nodes, 0x0d, 0x20 in some (last node?)**
498	1		**Unknown (Empty value)**
499	1		**Unknown (Empty value)**

Offset	Size	Value	Description
500	2		Type indicator **0x85 0x85 is used for ???**
502	2		**Unknown (Node identifier?)**
504	4		Back pointer must match the back pointer that pointed to this node
508	4		A weak CRC32 of the first 500 bytes of the index node

7. The density list

The density list is used to maintain a list of the data allocation tables in order of density.

The list starts with the low-density (free) data allocation tables?

According to [MS-PST] there is only a single density list at offset 0x4200.

Only found in newer PST and OST files?

7.1. The 32-bit density list

The 32-bit density list is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	1		Flags See section: The density list flags
1	1		Number of list entries
2	2	0	**Unknown (Padding)**
4	4		Next page index
8	480		Density list entries See section: The density list entry Maximum of 480 the remaining values are zeroed
488	12		**Unknown**
Footer 12 bytes of size			
500	1	0x86	Page type See section: Page types
501	1	0x86	Copy of page type
502	2		**Unknown (Signature)**
504	4		**Unknown (The back pointer)**
508	4		A weak CRC32 of the first 500 bytes of the density list

7.2. The 64-bit density list

The 64-bit density list is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	1		Flags See section: The density list flags
1	1		Number of list entries
2	2	0	**Unknown (Padding)**
4	4		Next page index
8	476		Density list entries See section: The density list entry Maximum of 476 the remaining values are zeroed
484	12		**Unknown**
Footer 16 bytes of size			
496	1	0x86	Page type See section: Page types
497	1	0x86	Copy of page type
498	2		**Unknown (Signature)**
500	4		A weak CRC32 of the first 496 bytes of the density list
504	8		**Unknown (The back pointer)**

7.3. The 64-bit 4k page density list

Seen at offset 0x21000

Offset	Size	Value	Description
			Unknown
Footer 24 bytes of size			
4072	1	0x86	Page type See section: Page types
4073	1	0x86	Copy of page type
4074	2		**Unknown (Signature)**
4076	4		A weak CRC32 of the first 4072 bytes of the density list
4080	8		Back pointer The value contains the allocation table offset
4088	8		**Unknown**

7.4. The density list flags

Value	Identifier	Description
0x01	DFL_BACKFILL_COMPLETE	Set if no backfill operation is in progress This flag has influence on the meaning of the next page index value

7.5. The density list entry

The density list entry is 32-bit of size and consists of:

Offset	Size	Value	Description
0.0	20 bits		The page number of the data allocation table
2.4	12 bits		Number of free entries in the data allocation table

The corresponding file offset of a page number is determined by the following calculation:

```
offset = page number x page size (512)
```

8. Blocks

[MS-PST] defines a common structure for storing raw data, data arrays and local descriptors as the block. Blocks should be 64 byte aligned, which is the granularity of the data allocation map. Blocks contain other data types like the local descriptor list, array and table.

The the maximum size of a block is 8192 bytes. The 64-bit 4k page block has a likely maximum size of 8 x 8192 = 65536 bytes, since the page size is 8 times larger than the non 4k page block variant (8 x 512 = 4096).

In a 32-bit PST the commonly used largest block size is 8192 - 12 = 8180, so the maximum of the block data size is the block size - size of the block footer.

The block footer contains a back pointer which refers back to the (file) offset index nodes. This value can be used to validate the integrity of the file.

8.1. The 32-bit block

The 32-bit block is variable of size in 64 byte increments and consists of:

Offset	Size	Value	Description
0	...		Block data
...	...		**Unknown (Padding)** **values are not always 0**

Offset	Size	Value	Description
Footer 12 bytes of size			
…	2		Block data size
…	2		Signature
…	4		Back pointer
…	4		A weak CRC32 of the block data Not including the padding

8.2. The 64-bit block

The 64-bit block is variable of size in 64 byte increments and consists of:

Offset	Size	Value	Description
0	…		Block data
…	…		**Unknown (Padding)** **values are not always 0**
Footer 16 bytes of size			
…	2		Block data size
…	2		Signature
…	4		A weak CRC32 of the block data Not including the padding
…	8		Back pointer

8.3. The 64-bit 4k page block

The 64-bit 4k page block is variable of size in 512 byte increments and consists of:

Offset	Size	Value	Description
0	…		Block data
…	…		**Unknown (Padding)**
Footer 24 bytes of size			
…	2		Block data size
…	2		Signature
…	4		A weak CRC32 of the block data Not including the padding
…	8		Back pointer
…	2	2	**Unknown**

Offset	Size	Value	Description
…	2		Uncompressed block data size **Or could this value be 32-bit?**
…	4		**Unknown (empty values)**

In the 64-bit 4k page the block data can be compressed. The compression method used is deflate/RFC1951.

If the block is compressed note that the size in the corresponding 64-bit (file) offset index entry contains the compressed block data size and not that of the uncompressed data.

TODO: confirm encryption comes before decompression, for now it seems that compressed files are not encrypted. Can an array be compressed if so how.

8.4. Block types

Value	Identifier	Description
	Data block	The raw data block has the 'external' bit set on the (file) offset index identifier. In a raw data block the block data value can be encrypted.
0x01	XBLOCK	The array block is used to store raw data greater than 8176 bytes. [MS-PST] refers to this structure as the XBLOCK which is a single level array. See section: The array for the contents of the block data value.
0x01	XXBLOCK	The array block is used to store raw data greater than **2\^16 x 8176** bytes. [MS-PST] refers to this structure as the XXBLOCK which is a two level array. See section: The array for the contents of the block data value.
0x02	SLBLOCK SIBLOCK	The local descriptors See section: The local descriptors for the contents of the block data value.

9. The array

The array is used when a (file) offset index identifier contains more data than can fit in a single (descriptor) data block. The array contains a set of (file) offset index identifiers.

The array is used for both table as for item value data.

The total data size should equal the sum of all the (file) offset index entry sizes referenced by the array.

The data of the individual array entries should be concatenated to each other in order.

According to [MS-PST] the maximum level of indirection is 2.

9.1. The 32-bit array

The 32-bit array is variable of size and consists of:

Offset	Size	Value	Description
0	1	0x01	The array signature (or block type)
1	1	0x01 0x02	The array (indirection) level 1 being the lowest level
2	2		The number of array entries
4	4		The total data size of the array entries
8	(number of entries x 4)		4 byte array entries containing (file) offset index identifiers

9.2. The 64-bit array

The 64-bit array is variable of size and consists of:

Offset	Size	Value	Description
0	1	0x01	The array signature (or block type)
1	1		The array (indirection) level 1 being the lowest level
2	2		The number of array entries
4	4		The total (uncompressed) data size of the array entries
8	(number of entries x 8)		8 byte array entries containing (file) offset index identifiers

The format for the 64-bit array for 512 and 4k page sizes is the same.

The total data size of the array entries is based on the uncompressed size of the data blocks.

10. The local descriptors

The local descriptors identifier in the descriptor index b-tree leaf node entry refers to a (file) offset index entry which contains the file offset and data size of the local descriptors nodes.

The local descriptors nodes make up a tree, that most of the time consists of only one level, therefore it was initially considered as a local descriptor list.

According to [MS-PST] the maximum level of indirection is 0x01.

10.1. The 32-bit local descriptors

The local descriptors contain descriptor (file) offset mappings for table data. The 32-bit local

descriptors are variable in size.

Offset	Size	Value	Description
0	1	0x02	The signature (or block type)
1	1		The node (indirection) level
2	2		The number of entries
4	(number of entries x entry size)		The entries

10.1.1. The 32-bit local descriptor branch nodes

The 32-bit local descriptors branch nodes have a level other than 0x00. An entry within the node is 8 bytes of size.

Offset	Size	Value	Description
0	4		The descriptor identifier
4	4		The (file) offset index identifier of the sub node.

The lower bit in data identifier should be cleared before searching the value in the (file) offset index.

If an attachment identifier is stored in a local descriptor branch node the corresponding the (file) offset index identifier of the data is in the sub node of the local descriptor branch node.

10.1.2. The 32-bit local descriptors leaf node

The 32-bit local descriptors leaf node has a level of 0x00. An entry within the node is 12 bytes of size.

Offset	Size	Value	Description
0	4		The descriptor identifier
4	4		The (file) offset index identifier of the data
8	4		The (file) offset index identifier of the local descriptors

The lower bit in data identifier should be cleared before searching the value in the (file) offset index.

The (file) offset index identifier of the local descriptors are mainly used in email items for attachments. It refers to the local descriptors of the attachment item.

10.2. The 64-bit local descriptors

The local descriptors contain descriptor (file) offset mappings for table data. The 64-bit local descriptors are variable in size.

Offset	Size	Value	Description
0	1	0x02	The signature
1	1		The node (indirection) level
2	2		The number of entries
4	4	0	**Unknown (Padding)**
8	(number of entries x entry size)		The entries

The format for the 64-bit local descriptors for 512 and 4k page sizes is the same.

10.2.1. The 64-bit local descriptor branch nodes

The 64-bit local descriptors branch nodes have a level other than 0x00. An entry within the node is 16 bytes of size.

Offset	Size	Value	Description
0	8		The descriptor identifier Identifier of type node identifier, only 32-bit are used See section: Index identifier
8	8		The (file) offset index identifier of the sub node.

The lower bit in data identifier should be cleared before searching the value in the (file) offset index.

If an attachment identifier is stored in a local descriptor branch node the corresponding the (file) offset index identifier of the data is in the sub node of the local descriptor branch node.

10.2.2. The 64-bit local descriptors leaf node

The 64-bit local descriptors leaf node has a level of 0x00. An entry within the node is 24 bytes of size.

Offset	Size	Value	Description
0	8		The descriptor identifier Identifier of type node identifier, only 32-bit are used See section: Index identifier
8	8		The (file) offset index identifier of the data
16	8		The (file) offset index identifier of the local descriptor

The lower bit in data identifier should be cleared before searching the value in the (file) offset index.

The (file) offset index identifier of the local descriptor is mainly used in email items for attachments. It refers to the local descriptors of the attachment item.

10.3. The 64-bit 4k page local descriptors

The 64-bit 4k page local descriptors are in the same format as the 64-bit local descriptors.

11. The table

The table contains entries which make up the items like email or contact. If the encryption type was set in the file header data the entire table is encrypted. Note that the not encrypted flag in the offset identifier can overwrite the table being encrypted.

The data identifier in the descriptor index b-tree leaf node entry refers to a (file) offset index entry which contains the file offset and data size of the table.

The table is made up of one or more table blocks. These table blocks can be stored in a table array.

11.1. The table block

The table block is variable of size and consists of:

- table block header
- table block values
- table block index

11.1.1. Table block header

The table block header is 16 bytes of size and consists of:

Offset	Size	Value	Description
0	2		The table block index offset
2	1	0xec	Signature
3	1		The table type (or client signature) See section: The table types
4	4		The table value reference
8	4		**Unknown (Fill level array)** **(8 x 4 bits entry)**

Only the first table block in a table array contains a table header.

According to [MS-PST] the fill level array only applies to the 8 first table blocks of the table array. The table block header of 2nd to 8th table array entries is 2 bytes of size and consists of:

Offset	Size	Value	Description
0	2		The table index offset

This header is repeated every table array entry not needed to contain a fill level array.

The table header of the 9th table array entry is 66 bytes of size and consists of:

Offset	Size	Value	Description
0	2		The table index offset
2	64		**Unknown (Fill level array)** **(128 x 4 bits entry)**

This header is repeated every 128 table array entries, e.g. in table array entry 137.

According to [MS-PST] the fill level array entries for non existing table array entries should be set to 0.

11.1.2. The table types

The following table types are currently known.

Table type	Description	Features
0x6c	6c table	Has GUID record entry identifiers Has table specific table header Has b5 table header Has a GUID table values array
0x7c	7c table	(Table context) Has MAPI property (based) record entry identifiers Has table specific table header Has b5 table header Has column definitions array Has a table values array
0x8c	8c table	Has MAPI property (based) record entry identifiers Has b5 table header
0x9c	9c table	Has GUID record entry identifiers Has table specific table header Has b5 table header
0xa5	a5 table	Has MAPI property (based) record entry identifiers
0xac	ac table	Has MAPI property (based) record entry identifiers Has table specific table header Has b5 table header Has column definitions array Has a table values array
0xb5	b5 table header	(B-Tree on heap)
0xbc	bc table	(Property context) Has MAPI property (based) record entry identifiers Has b5 table header
0xcc	cc table	**Unknown**

11.1.3. The table fill level

Value	Identifier	Description
0x0	FILL_LEVEL_EMPTY	value >= 3584 bytes free or non-existent data block
0x1	FILL_LEVEL_1	2560 >= value > 3584 bytes free
0x2	FILL_LEVEL_2	2048 >= value > 2560 bytes free
0x3	FILL_LEVEL_3	1792 >= value > 2048 bytes free
0x4	FILL_LEVEL_4	1536 >= value > 1792 bytes free
0x5	FILL_LEVEL_5	1280 >= value > 1536 bytes free
0x6	FILL_LEVEL_6	1024 >= value > 1280 bytes free
0x7	FILL_LEVEL_7	786 >= value > 1024 bytes free
0x8	FILL_LEVEL_8	512 >= value > 786 bytes free
0x9	FILL_LEVEL_9	256 >= value > 512 bytes free
0xa	FILL_LEVEL_10	128 >= value > 256 bytes free
0xb	FILL_LEVEL_11	64 >= value > 128 bytes free
0xc	FILL_LEVEL_12	32 >= value > 64 bytes free
0xd	FILL_LEVEL_13	16 >= value > 32 bytes free
0xe	FILL_LEVEL_14	8 >= value > 16 bytes free
0xf	FILL_LEVEL_FULL	value < 8 bytes free

11.1.4. The table block index

The table block index is variable of size and consists of:

Offset	Size	Value	Description
0	2		The number of index offsets
2	2		The number of unused offsets items
4	(number of items + 1) x 2		Array of index offsets An index offset contains the offset of the table block value. The index offset is relative to the start of the table block.

Note that:

- the first index offsets is referred to as number 1;
- the index offsets are stored in order;
- the last index offset does not have to match the table block index offset;
- the number of index offsets can be 0.

11.2. The table value reference

The table value reference is formatted in different ways, it can point to data either in within the table block or in some other block.

11.2.1. 32-bit and 64-bit table value reference

The table value reference is 32-bit of size and consists of:

Offset	Size	Value	Description
0.0	5 bits		The value reference type See section: Node identifier types
0.5	11 bits		The value reference index
2.0	16 bits		The value reference array index

- internal table value references have the all the low order 4 bits zero e.g. 0x0020, the value needs to be right shifted by 5 bits, e.g. 0x0001. This value is the first entry in the the table index (starts at 1), so it points to a table index value offset e.g. 12 (0xc). for internal table values references the high order 16 bits are used to indicate which table array entry should be used, e.g. a high order value of 1 points to the second table array entry;
- external table value references have some of the low order 4 bits set (**and the value reference array index is 0**). They are descriptor list identifiers that refer to another location of data.

TODO: Check with [MS-PST] p 56

11.2.2. 64-bit 4k page table value reference

The table value reference is 32-bit of size and consists of:

Offset	Size	Value	Description
0.0	5 bits		The value reference type See section: Node identifier types
0.5	14 bits		The value reference index
2.3	13 bits		The value reference array index

11.2.3. Internal table value reference

An internal table value reference refers to the first table index value pair that contain the table values descriptor.

The internal table value reference for:

Table type	Use of internal table value reference
0x6c	points to table specific 6c table header
0x7c	points to table specific 7c table header
0x8c	points to b5 table header
0x9c	points to table specific 9c table header
0xa5	points to record entries
0xac	points to table specific ac table header
0xbc	points to b5 table header

11.3. The b5 table header

The b5 table header is used in all table types except the a5 table. It contains information how the record entries are formatted. It consists of 8 bytes:

Offset	Size	Value	Description
0	1	0xb5	Table header type
1	1		The size of the record entry identifier Either 2, 4, 8 or 16
2	1		The size of the record entry value 0 > value >= 32
3	1		The level of record entries
4	4		record entries reference

The record entry index reference refers to the table index value pair that points to record entries. If the record entries reference is zero there are no record entries.

The level of the record entries is used to distribute the record entries over multiple table values. Intermediate level record entries are variable of size and consist of:

Offset	Size	Value	Description
0	…		record entry identifier (key)
…	4		record entries sub reference

Where leaf level record entries are variable of size and consist of:

Offset	Size	Value	Description
0	…		record entry identifier (key)
…	4		record entry data

The size of an individual record entry is the combination of the record entry identifier and value size.

The b5 table header values differs for different tables:

Table type	record entry identifier size	record entry value size	record entry size
0x6c	16	2	18
0x7c	4	2	6
0x7c	4	4	8
0x8c	8	4	12
0x9c	16	4	20
0xac	4	4	8
0xbc	2	6	8

The individual table sections provide more information about the values in the record entries.

11.4. The 6c table

The bc table has table values that contain:

- a b5 table header
- a 6c table header
- record entries that contain contain GUID descriptor values and the value array information
- value array (table) entries that contain the item value information

11.4.1. The 6c table header

The 6c table header consists of 8 bytes:

Offset	Size	Value	Description
0	4		The b5 table header index reference
4	4		Values array entries index reference

11.4.2. The b5 table header entry

The 6c table uses the b5 table header with a record entry identifier size of 16 and a record entry value size of 2. The record entries reference refers to the record entries. If the record entries reference is zero there are no record entries.

11.4.3. The record entries

A b5 table header with a record entry identifier size of 16 and a record entry value size of 2 refers to a specific type of record entry. This type of record entry consists of 18 bytes:

Offset	Size	Value	Description
0	16		A GUID
16	2		**Unknown** **First part of the value in PRQ_ID_SECURE4**

11.5. The 7c table

The bc table has table values that contain:

- a b5 table header
- a 7c table header
  - 7c column definitions that contain the item type information
- record entries that contain the value array information
- value array (table) entries that contain the item value information

11.5.1. The 7c table header

The 7c table header consists of 22 bytes:

Offset	Size	Value	Description
0	1	0x7c	Table header type
1	1		The number of column definitions
2	2		values array entry end offset 32-bit values End offset of the 4 or 8 byte values
4	2		values array entry end offset 16-bit values End offset of the 2 byte values

Offset	Size	Value	Description
6	2		values array entry end offset 8-bit values End offset of the 1 byte values
8	2		values array entry end offset cell existence block **(The values array entry size)**
10	4		The b5 table header index reference
14	4		Values array entries index reference
18	4		**Unknown (hidIndex)** Deprecated according to [MS-PST] and should be set to 0

If the b5 header table index reference is zero the table should not contain any record entries. If the value array entries index reference is zero the table does not contain any value array entries.

The record entries contain references to the table value array entries. So if the table contains no values the value array should be empty.

In some tables the b5 table header index reference contains a references to a b5 table header with an empty record entries reference. The value array entries index reference in the 7c table header is also empty.

It is possible for the table to have table header entries but no values array entries. **The reverse is unknown.**

11.5.2. The 7c column definition

The remaining data in the 7c table header contains multiple column definitions. The column definitions describe the format of the data in the values array entries. The 7c column definition consist of 8 bytes:

Offset	Size	Value	Description
0	2		The record entry value type
2	2		The record entry type
4	2		The values array entry offset
6	1		The values array entry size
7	1		The values array entry number (0 represents the first entry) **Cell existence bitmap index**

If the table contains values array entries the values array entry offset contains the offset of the value in the value array (table) entries.

In case of a value reference the actual value is found by reading the value size number of bytes from the value array entries at the specified value array entries offset. A value array entries offset of 0 points to the beginning of the value array entries.

11.5.3. The b5 table header entry

The 7c table uses the b5 table header with a record entry identifier size of 4 and a record entry value size of 2 or 4. The record entries reference refers to the record entries. If the record entries reference is zero there are no record entries.

11.5.4. The record entries

The record entries branch

The record entries branch has a record entries level value of 1 (and probably higher). The initial tables entries level is specified in the b5 table header. A record entry branch consists of 8 bytes:

Offset	Size	Value	Description
0	4	The first value in the lower level record entry array	4

The record entry branch contains a reference to lower level record entries.

The record entries leaf

The record entries leaf has a record entries level value of 0. The initial tables entries level is specified in the b5 table header.

A b5 table header with a record entry identifier size of 4 and a record entry value size of 2 refers to a record entry consists of 6 bytes:

Offset	Size	Value	Description
0	4		The first value in the value array
4	2		Value array number

A b5 table header with a record entry identifier size of 4 and a record entry value size of 4 refers to a record entry consists of 8 bytes:

Offset	Size	Value	Description
0	4		The first value in the value array
4	4		Value array number

11.5.5. The values array entries

The values array entries contain item entries values. The 7c header entries define the format of the entry/value data within an array entry. The value size and value array entries offset in the 7c header entries refer to the item value in the value arrays.

The value array consist of multiple values of different sizes.

Offset	Size	Value	Description
0	…		The 4 and 8 byte values
…	…		The 2 byte values
…	…		The 1 byte values
…	…		The cell existence block bitmap Every bit represent if a value (or column) exists

For record entry value types that fit into the specified size the record entry value is used directly, i.e. 32-bit, like Integer 32-bit signed (0x0003) or 64-bit, like Filetime (0x0040). Otherwise, the record entry value is a value reference, which is either a descriptor list identifier, or a table index reference. If the record entry value is 0 the value is empty. Unlike the bc table the 7c table does store values smaller than 32-bit in lesser number of bytes.

If a values array reference is an external reference and the values array is stored in a data array there is additional padding at the end of the last value array in a certain data array block. If the data in the data array is assumed continuous this causes a misalignment for the value array in the next data array block. The value array entry identifier in the record entries can be used to realign.

11.6. The 8c table

The 8c table has table values that contain:

- a b5 table header
- record entries that contain identifier to descriptor mappings

11.6.1. The b5 table header entry

The 8c table uses the b5 table header with a record entry identifier size of 8 and a record entry value size of 4. The record entries reference refers to the record entries. If the record entries reference is zero there are no record entries.

11.6.2. The record entries

A b5 table header with a record entry identifier size of 16 and a record entry value size of 2 refers to a specific type of record entry. This type of record entry consists of 18 bytes:

Offset	Size	Value	Description
0	8		**Unknown (Identifier)** **Similar to the value in PRQ_ID_SECURE4**
8	4		Descriptor identifier **with the last 4 bits masked as zero**

11.7. The 9c table

The 9c table has table values that contain:

- a b5 table header
- a 9c table header
- record entries that contain GUID descriptor values

11.7.1. The 9c table header

The ac table header consists of 4 bytes:

Offset	Size	Value	Description
0	4		b5 table header index reference

11.7.2. The b5 table header entry

The 9c table uses the b5 table header with a record entry identifier size of 16 and a record entry value size of 4. The record entries reference refers to the record entries. If the record entries reference is zero there are no record entries.

11.7.3. The record entries

A b5 table header with a record entry identifier size of 16 and a record entry value size of 4 refers to a specific type of record entry. This type of record entry consists of 20 bytes:

Offset	Size	Value	Description
0	16		A GUID
16	4		A descriptor identifier

11.8. The a5 table

The a5 table has table values that contain:

- record entries that contain record entry values

The a5 table is used by the ac column definitions as an array of record entry values.

The internal table value reference for the a5 table is 0.

If the a5 table is empty it signifies NULL values;

11.9. The ac table

The ac table has table values that contain:

- a b5 table header
- a ac table header
- ac column definitions that contain the item type information
    - a5 tables containing the actual record entry values
- record entries that contain the value array information
- value array (table) entries that contain the item value information

11.9.1. The ac table header

The ac table header consists of 40 bytes:

Offset	Size	Value	Description
0	1	"\xac"	Table header type
1	1		**Unknown (Empty value)**
2	2		values array entry end offset 32-bit values End offset of the 4 or 8 byte values
4	2		values array entry end offset 16-bit values End offset of the 2 byte values
6	2		values array entry end offset 8-bit values End offset of the 1 byte values
8	2		values array entry end offset cell existence block **(The values array entry size)**
10	4		B5 table header index reference
14	4		Values array entry reference
18	4		**Unknown (Empty value)**
22	2		Number of column definitions
24	4		column definitions reference
28	8		**Unknown (Empty value)**
36	4		**Unknown value (Weak CRC?)**

11.9.2. The ac column definition

The column definitions reference refers to the ac column definitions. The column definitions describe the format of the data in the values array entries. The ac column definition consist of 16 bytes:

Offset	Size	Value	Description
0	2		The record entry value type
2	2		The record entry type
4	2		The values array entry offset

Offset	Size	Value	Description
6	2		The values array entry size
8	2		The values array entry number (0 represents the first entry)
10	2		**Unknown (Empty value)**
12	4		The descriptor identifier of the record entry values table (a5 table)

If the table contains values array entries the values array entry offset contains the offset of the value in the value array (table) entries.

In case of a value reference the actual value is found by reading the value size number of bytes from the value array entries at the specified value array entries offset. A value array entries offset of 0 points to the beginning of the value array entries.

11.9.3. The b5 table header entry

The ac table uses the b5 table header with a record entry identifier size of 4 and a record entry value size of 4. The record entries reference refers to the record entries. If the record entries reference is zero there are no record entries.

It might be that the 4 + 2 variant like for the 7c table is also possible for the ac table.

11.9.4. The record entries

The record entries branch

The record entries branch has a record entries level value of 1 (**and probably higher**). The initial tables entries level is specified in the b5 table header. A record entry branch consists of 8 bytes:

Offset	Size	Value	Description
0	4		The first value in the lower level record entry array
4	4		The value reference of the lower level record entry array

The record entry branch contains a reference to lower level record entries.

The record entries leaf The record entries leaf has a record entries level value of 0. The initial tables entries level is specified in the b5 table header.

A b5 table header with a record entry identifier size of 4 and a record entry value size of 4 refers to a specific type of record entry. This type of record entry consists of 8 bytes:

Offset	Size	Value	Description
0	4		The first value in the value array
4	4		Value array number

11.9.5. The values array entries

The values array entries contain item entries values. The ac header entries define the format of the entry/value data within an array entry. The value size and value array entries offset in the ac header entries refer to the item value in the value arrays.

The value array consist of multiple values of different sizes.

Offset	Size	Value	Description
0	…		The 4 and 8 byte values
…	…		The 2 byte values
…	…		The 1 byte values
…	…		The cell existence block bitmap Every bit represent if a value (or column) exists

For record entry value types that fit into the specified size the record entry value is used directly, i.e. 32-bit, like Integer 32-bit signed (0x0003) or 64-bit, like Filetime (0x0040). Otherwise, the record entry value is a value reference, which is either a descriptor list identifier, or a table index reference. If the record entry value is 0 the value is empty. Unlike the bc table the ac table does store values smaller than 32-bit in lesser number of bytes.

Some column definitions have a descriptor identifier of the record entry values table. This descriptor identifier refers to an a5 table which contains an array of record entry values. In this case the value in the values array actually contains an item index of the a5 table.

If a values array reference is an external reference and the values array is stored in a data array there is additional padding at the end of the last value array in a certain data array block. If the data in the data array is assumed continuous this causes a misalignment for the value array in the next data array block. The value array entry identifier in the record entries can be used to realign.

11.10. The bc table

The bc table has table values that contain:

- a b5 table header
- record entries that contain the item type/value information
- record entry value data

11.10.1. The b5 table header entry

The bc table uses the b5 table header with a record entry identifier size of 2 and a record entry value size of 6. The record entries reference refers to the record entries. If the record entries reference is zero there are no record entries.

11.10.2. The record entries

The record entries branch

The record entries branch has a record entries level value of 1 (and probably higher). The initial tables entries level is specified in the b5 table header. A record entry branch consists of 6 bytes:

Offset	Size	Value	Description
0	2		The first value in the lower level record entry array
2	4		The value reference of the lower level record entry array

The record entry branch contains a reference to lower level record entries.

The record entries leaf

The record entries leaf has a record entries level value of 0. The initial tables entries level is specified in the b5 table header.

The record entries in the bc table contain item entries. This type of record entry consists of 8 bytes:

Offset	Size	Value	Description
0	2		The record entry type
2	2		The record entry value type
4	4		The record entry value or value reference

For record entry value types that fit into 32-bit, like Integer 16-bit signed (0x0002), Integer 32-bit signed (0x0003), Boolean (0x000b), the record entry value is used directly. Otherwise, the record entry value is a value reference, which is either a descriptor list identifier, or a table index reference. If the record entry value is 0 the value is empty.

11.11. The cc table

According to [MS-PST] there should be a cc table, however it is undocumented and has not yet been spotted in the wild.

11.12. The item and item value types

The item and item value types are defined in the MAPI definitions document.

The item types are also referred to as the MAPI Property Names/Identifiers (PR_) or columns by scanpst. The item value types are also referred to as the MAPI Property (Data) Types (PT).

12. The PFF items

The PFF items are stored in record entries. Different tables make up different PFF items.

12.1. Internal nodes

Several of the PFF items have a predefined node identifier.

Value	Identifier	Description
33 (0x21)	NID_MESSAGE_STORE	The message store Consists of a bc table
97 (0x61)	NID_NAME_TO_ID_MAP	The name-to-id-map Consists of a bc table
161 (0xa1)	NID_NORMAL_FOLDER_TEMPLATE	The folder template
193 (0xc1)	NID_SEARCH_FOLDER_TEMPLATE	The search folder template
290 (0x122)	NID_ROOT_FOLDER	The root folder Consists of a bc table **Note that this actually is a folder node type (0x02)**
481 (0x1e1)	NID_SEARCH_MANAGEMENT_QUEUE	Pending search-related update queue **Consists of an empty descriptor**
513 (0x201)	NID_SEARCH_ACTIVITY_LIST	Active searches list **Consists of a list of some kind**
577 (0x241)	NID_RESERVED1	**Unknown (Reserved)**
609 (0x261)	NID_SEARCH_DOMAIN_OBJECT	Search criteria list **Consists of a list of some kind**
641 (0x281)	NID_SEARCH_GATHERER_QUEUE	Search gatherer queue **Consists of an empty descriptor**
673 (0x2a1)	NID_SEARCH_GATHERER_DESCRIPTOR	Search gatherer descriptor **Consists of (yet) unknown data**
737 (0x2e1)	NID_RESERVED2	**Unknown (Reserved)**
769 (0x301)	NID_RESERVED3	**Unknown (Reserved)**

Value	Identifier	Description
801 (0x321)	NID_SEARCH_GATHERER_FOLDER_QUEUE	Search gatherer folder queue **Consists of an empty descriptor**
2049 (0x801)		**Unknown (found in OST)** Consists of a 6c table
2081 (0x821)		**Unknown (found in OST)** Consists of a 8c table
2113 (0x841)		**Unknown (found in OST)** Consists of a 7c table
3073 (0xc01)		**Unknown (found in PST, OST)** Consists of a 9c table

12.2. The message store

The descriptor index identifier 33 (0x21) refers to the message store.

The message store is a bc table which can contain:

- The display name: "Personal Folders"
- Valid folder mask
- Password checksum

The message store contains several entry identifiers of Outlook special folders. These are:

Folder	Entry identifier property
Outbox folder	PidTagIpmOutboxEntryId (PR_IPM_OUTBOX_ENTRYID)
Deleted Items folder	PidTagIpmWastebasketEntryId (PR_IPM_WASTEBASKET_ENTRYID)
Sent Items folder	PidTagIpmSentMailEntryId (PR_IPM_SENTMAIL_ENTRYID)
IPM root folder	PidTagIpmSubtreeEntryId (PR_IPM_SUBTREE_ENTRYID)
Search-results root folder	PidTagFinderEntryId (PR_FINDER_ENTRYID)
Common views root folder	PidTagCommonViewsEntryId (PR_COMMON_VIEWS_ENTRYID)
Personal views root folder	PidTagViewsEntryId (PR_VIEWS_ENTRYID)

Folder	Entry identifier property
Contacts root folder	PidTagIpmContactEntryId (PR_IPM_CONTACT_ENTRYID)
Drafts root folder	PidTagIpmDraftsEntryId (PR_IPM_DRAFTS_ENTRYID)
Journal root folder	PidTagIpmJournalEntryId (PR_IPM_JOURNAL_ENTRYID)
Calendar root folder	PidTagIpmAppointmentEntryId (PR_IPM_APPOINTMENT_ENTRYID)
Notes root folder	PidTagIpmNoteEntryId (PR_IPM_NOTE_ENTRYID)
Tasks root folder	PidTagIpmTaskEntryId (PR_IPM_TASK_ENTRYID)

NOTE Some PFF files do not contain a message store.

12.3. The name-to-id map

The descriptor index identifier 97 (0x61) refers to the the name-to-id map.

The name-to-id map is a bc table which contains the following entries:

- 0x0001 (Name-to-ID Map Number of Validation Entries)
- 0x0002 (Name-to-ID Map Class identifiers)
- 0x0003 (Name-to-ID Map Entries)
- 0x0004 (Name-to-ID Map Strings)
- 0x1000 and up (Name-to-ID Map Validation Entries)

The entry 0x0002 (Name-to-ID Map Class Identifiers) is of type 0x0102 (Binary data) and contains an array of class identifiers (CLSID).

The entry 0x0003 (Name-to-ID Map Entries) is of type 0x0102 (Binary data) and contains an array of name-to-id map entries. An name-to-id map entry consist of 8 bytes.

Offset	Size	Value	Description
0	4		The name-to-id map entry value or value reference
4	2		The name-to-id map entry type
6	2		The name-to-id map entry number

The lowest bit in the name-to-id map entry type signifies where to find the name-to-id map value.

- If set it contains an offset into the 0x0004 (Name-to-ID Map Strings) array. This type corresponds to MAPI MNID_STRING;

- If not set it contains the entry type to which the name-to-id is mapped. This type corresponds to MAPI MNID_ID.

Type	Identifier	Description
0x0000 0x0001	NAMEID_GUID_NONE	No class
0x0002 0x0003	NAMEID_GUID_MAPI	The name-to-id map entry type refers to the class MAPI (PS_MAPI 00020328-0000-0000-c000-000000000046)
0x0004 0x0005	NAMEID_GUID_PUBLIC_STRINGS	The name-to-id map entry type refers to the class Public stings (PS_PUBLIC_STRINGS: 00020329-0000-0000-c000-000000000046)

The remaining name-to-id map entry type value refers to a value in the class identifier array:

```
index number = ( type / 2 ) - 3
```

E.g. the value 0x0006 or 0x0007 refer to the first entry (entry: 0) in the class identifier array.

The correspondent item type is the name-to-id map number + 0x8000.

The entry 0x0004 (Name-to-ID Map Strings) is of type 0x0102 (Binary data) and contains an array of strings. An individual string consists of:

Offset	Size	Value	Description
0	4		The number of bytes in the string
4	…		The string in ASCII or Unicode without the end of string character (NUL-character)

NOTE The Name-to-ID Map Strings can be empty.

Most of the time the Name-to-ID Map Strings are in Unicode (UTF-16) however sometimes the string consists of an ASCII string containing. Until now only ASCII strings containing a MAPI property identifier string (PR_) have been found. In particular in relation with a BlackBerry RIM server properties (PR_RIM_). Note that the last byte in such ASCII strings can be a 0 byte.

Could this be a mnemonic way of mapping MAPI identifiers?

The entries s0x1000 and up (Name-to-ID Map Validation Entries) contain values similar to those in the entry 0x0003 (Name-to-ID Map Entries). Except that these are used for validation.

Offset	Size	Value	Description
0	4		The name-to-id map entry validation value
4	2		The name-to-id map entry type
6	2		The name-to-id map entry number

The lowest bit in the name-to-id map entry type signifies where to find the name-to-id map validation value.

- If set it contains a weak CRC32 of the string in the 0x0004 (Name-to-ID Map Strings) array;
- If not set it contains a duplicate of the value in the 0x0003 (Name-to-ID Map Entries).

NOTE Some PFF files do not contain a name-to-id map.

12.4. The root folder and folder items

The descriptor index identifier 290 (0x112) refers to the the root folder item.

The descriptor index entry of the root item refers to itself as its parent.

The child items can be found by the parent descriptor identifier in the descriptor index entry. The descriptor index entries that are not part of the item hierarchy should not contain parent identifiers (parent identifies of 0).

The folder item can contain:

- Display Name
- Number of content items
- Number of unread content items
- Has sub folders
- Associate content count

The number of content items in a folder is made up from the item count and associated item count.

12.4.1. Inbox special folders

The Inbox folder contains several entry identifiers of Outlook special folders. These are:

Folder	Entry identifier property
Calendar	PidTagIpmAppointmentEntryId (PR_IPM_APPOINTMENT_ENTRYID)
Contacts	PidTagIpmContactEntryId (PR_IPM_CONTACT_ENTRYID)
Journal	PidTagIpmJournalEntryId (PR_IPM_JOURNAL_ENTRYID)
Notes	PidTagIpmNoteEntryId (PR_IPM_NOTE_ENTRYID)
Tasks	PidTagIpmTaskEntryId (PR_IPM_TASK_ENTRYID)

Folder	Entry identifier property
Drafts	PidTagIpmDraftsEntryId (PR_IPM_DRAFTS_ENTRYID)

Outbox special folders?

12.4.2. The related sub folders item

A folder descriptor index identifier + 11 (0x000b) refers to the related sub folders item, e.g. for the root folder this is 290 + 11 = 301. The related sub folders item consists of a 7c table.

The sub folders item can be used to determine which items in the folder are sub folders. The row identifier (PidTagLtpRowNid) value of each set contains the identifier of the sub folder item.

12.4.3. The related sub messages item

A folder descriptor index identifier + 12 (0x000c) refers to the related sub messages item, e.g. for the root folder this is 290 + 12 = 302. The related sub messages item consists of a 7c table.

The sub messages item can be used to determine which items in the folder are messages. The row identifier (PidTagLtpRowNid) value of each set contains the identifier of the sub message item.

12.4.4. The related sub associated contents item

A folder descriptor index identifier + 13 (0x000d refers to the related sub associated contents item, e.g. for the root folder this is 290 + 13 = 303. The related sub associated contents item consists of a 7c table.

The sub associated contents item can be used to determine which items in the folder are associated contents. The row identifier (PidTagLtpRowNid) value of each set contains the identifier of the sub associated contents item.

12.4.5. Note

```
Special purpose folder descriptor index identifier + 3 => empty descriptor
Special purpose folder descriptor index identifier + 4 => bc table

A folder descriptor index identifier + 18 => 8c table (Inbox, Drafts, Sync Issues,
Renamed By MAE)
Contains an number of elements similar to the number of messages in the corresponding
folder.

A folder descriptor index identifier + 19 => 8c table (CommonViews, Inbox, Calendar)

Calender folder descriptor index identifier + 23 => 7c table
```

12.4.6. Unknown 1718 sub item

```
When a folder contains X the local descriptors contains an entry 1718 (0x06b6). This
local descriptor refers to a 7c table which contains the Y item.

The Y contains multiple sets (1 per Z). It can contain the:

(Used by: CommonViews, OutlookReminder, To-Do Search, Tracked Mail Processing,
Shortcuts, Views, IPM SUBTREE, Deleted Items, Inbox, Sent Items, Calendar, Contacts,
Drafts, Journal, Junk E-mail, Notes, RSS Feeds, Conflicts, )

Perhaps a Folder View or sort sub item?

0x67f2 ( : Row identifier)
points to an entry type within the parent table
```

12.4.7. Note

```
Possible other sub items
1751
1784
```

12.5. The message item

The message item is a generic variant of other items, like e-mail, contact, appointment, etc.

The message item at least has:

- a message class containing: 'IPM'

Certain types of message items also can have the sub items:

- attachments
- recipients

Value	Identifier	Description
1682 (0x0692)	NID_TYPE_RECIPIENT_TABLE	Recipients item Consists of a 7c table
1649 (0x0671)	NID_TYPE_ATTACHMNET_TABLE	Attachments item Consists of a 7c table
	NID_TYPE_ATTACHMNET	Attachment item Consists of a bc table

Value	Identifier	Description
	NID_TYPE_LTP	Table value reference Consists of raw data

12.5.1. The attachments sub item

When an item contains attachments the local descriptors contains an entry 1649 (0x0671). This local descriptor refers to a 7c table which contains the attachments item.

The attachments item table contains multiple sets (1 per attachment). It can contain the attachments:

- size
- filename
- attachment method
- attachment item local descriptor

The local descriptor identifier refers to an entry in the local descriptors of the corresponding item. The list identifier of the local descriptor entry refers to the local descriptors of the attachment item.

The attachment sub item

The attachment can contain the attachments:

- size
- creation time
- modification time
- data object
- filename
- attachment method
- mime type
- rendering position

The Attachment data object (0x3701) can be of type Binary Data (0x0102) and Embedded Object (0x000d). The Embedded Object is used for bounced e-mails in which it contains an embedded PFF table. The Embedded Object can also contain other data like an OLE2 document.

Attachment method (0x3705) value	Attachment data object (0x3701) entry type	Attachment type
0x00000000	**Unknown**	**Unknown**
0x00000001	Binary Data (0x0102)	(Attached) Data (Separately attached data)

Attachment method (0x3705) value	Attachment data object (0x3701) entry type	Attachment type
0x00000002	(Attachment is stored externally)	Reference
0x00000003	**Unknown**	**Unknown**
0x00000004	**Unknown**	**Unknown**
0x00000005	Embedded Object (0x000d)	(Attached) Item (Embedded PFF item)
0x00000006	Embedded Object (0x000d)	(Attached) Data or item (Embedded OLE2 document)

[yellow-background]*Found empty attachment data object (0x3701) value but attachment has size 206. But contains entry 0x0e27 with binary data of size 100.

Found attachment with attachment method (0x3705) 0x0001 (ATTACH_BY_VALUE) without an attachment data object (0x3701) entry. Attachment has size 54, however Outlook shows base64 encoded data of a larger size. Parent e-mail contains 'partial message' entries. This seems to be a split MIME RFC message, where the data is stored in the message body of the attached e-mails.

Outlook cannot access the attachment of the attached e-mails.*

12.5.2. The recipients sub item

When a folder contains attachments the local descriptors contains an entry 1682 (0x0692). This local descriptor refers to a 7c table which contains the recipients items.

The recipients item contains multiple sets (1 per recipient). It can contain the recipient:

- type
- e-mail address
- search key
- messaging username

12.5.3. Note

```
Possible other sub items
1612
```

12.6. The appointment item

The appointment item can contain the values:

- message class: 'IPM.Appointment'

The appointment item also can have the sub items:

- attachments
- recipients

12.7. The contact item

The contact item can contain the values:

- message class: 'IPM.Contact'

12.8. The distribution list item

The distribution list item is also known as contact group item can contain the values:

- message class: 'IPM.DistList'
- 0x8054 (PidLidDistributionListOneOffMembers)
- 0x8055 (PidLidDistributionListMembers)

12.9. The e-mail item

The e-mail item can contain the values:

- message class: 'IPM.Note'
- transport headers
- plain text message body
- compressed RTF message body
- HTML message body

The e-mail item also can have the sub items:

- attachments
- recipients

12.10. The sticky note item

The task item can contain the values:

- message class: 'IPM.StickyNote'

12.11. The task item

The task item can contain the values:

- message class: 'IPM.Task'

12.12. The message manager (associated) item

The message manager (associated) item can contain the values:

- message class: 'IPM.MessageManager'

12.13. The migration status (associated) item

The migration status (associated) item can contain the values:

- message class: 'IPM.Microsoft.MigrationStatus'

12.14. The rule organizer (associated) item

The rule organizer (associated) item can contain the values:

- message class: 'IPM.RuleOrganizer'

12.15. The rule message (associated) item

The rule message (associated) item can contain the values:

- message class: 'IPM.Rule.Message'

12.16. The extended rule message (associated) item

The extended rule message (associated) item can contain the values:

- message class: 'IPM.ExtendedRule.Message'

12.17. The configuration RSS rule (associated) item

The configuration RSS rule (associated) item can contain the values:

- message class: 'IPM.Configuration.RssRule'

13. LZFu compression

The LZFu compression is used for RTF formatted data [ROTHMAN99].

Compressed LZFu data starts with a LZFu header

Offset	Size	Value	Description
0	4		Size of the compressed data including the following 12 bytes of the header
4	4		Size of the uncompressed data
8	4		Compression signature
12	4		A CRC32 of the compressed data.

The signature 0x75465a4c ("LZFu") that the data is compressed. The signature 0x414c454d ("MELA") that the data is uncompressed.

The CRC32 is similar to the standard CRC32 algorithm which is mentioned in RFC 1952 with a slight modification: the inversion (or xor with 0xffffffffL) before and after the CRC update is omitted. This inversion is applied in order to avoid a CRC weakness, which is that any number of leading or trailing zero bytes can be added or removed without the CRC detecting the change. For some reason, the compressed RTF CRC32 implementation is the weaker one, without this inversion. It is calculated on the compressed data bytes (excluding the LZFu header).

The compressed data is directly after the header. It consists of 8-unit chunks. Each chunk begins with a single flag byte. The bits within the flag byte are read in LSB order. Each bit in the byte flag is a flag for the corresponding unit in the chunk.

- 0 represent a 1 byte literal which should be copied as-is;
- 1 represent a 2 byte reference.

A 2 byte reference consists of:

Offset	Size	Value	Description
0	1.4		Reference offset into the LZ buffer
1.4	0.4		Reference size

The reference offsets represent offsets into the LZ buffer. The size of the reference offset allows for 4096 possible values, which is the size of the LZ buffer. The LZ buffer wraps around as it is filled with the decompressed data. The LZ buffer is preloaded with a common RTF header string (found in RTFLIB32.LIB). The string is represented as a C string of 207 bytes.

```
{\\rtf1\\ansi\\mac\\deff0\\deftab720{\\fonttbl;}{\\f0\\fnil \\froman \\fswiss
\\fmodern \\fscript \\fdecor MS Sans SerifSymbolArialTimes New
RomanCourier{\\colortbl\\red0\\green0\\blue0\n\r\\par
\\pard\\plain\\f0\\fs20\\b\\i\\u\\tab\\tx
```

The reference size is a 4 bit value that represents a value between 2 and 17. A reference size of 0 representing 2. Therefore the reference size needs to be corrected by 2.

The uncompressed size does not entail the 2 trailing zero bytes.

14. MacBinary encoding

PST and presumably OST files created by Microsoft Entourage (Outlook for MacOS) seem to use the MacBinary encoding to store attachment data.

Three MacBinary standards are known:

- version 1 (MacBinary)
- version 2 (MacBinary II)
- version 3 (MacBinary III)

The MacBinary format uses big-endian.

MacBinary data consists of:

Offset	Size	Value	Description
0	128		Header See section: MacBinary Header
128	(secondary header size)		Secondary header as of MacBinary II if set this value must be 128 byte aligned padded with zero values if necessary
…	(data fork size)		The data fork if set this value must be 128 byte aligned padded with zero values if necessary
…	(resource fork size)		The resource fork if set this value must be 128 byte aligned padded with zero values if necessary
…	(get info size)		get info as of MacBinary II

14.1. MacBinary header

The MacBinary header is 128 bytes in size and consists of:

Offset	Size	Value	Description
As of MacBinary			
0	1	0x00	**Unknown (Reserved for version byte)**
1	1		Filename length Minimum of 1, Maximum of 31
2	63		Filename Unused bytes are zeroed
65	4		The file type

Offset	Size	Value	Description
69	4		The file creator
73	1		Finder flags (bits 8-15) See section: Finder flags
74	1	0x00	**Unknown (Empty value)**
75	2		y coordinate The file's vertical position in the Finder window
77	2		x coordinate The file's horizontal position in the Finder window
79	2		Folder identifier The file's Finder window or folder identifier
81	1		Protected flag **0x01 ⇒ ?**
82	1	0x00	**Unknown (Empty value)**
83	4		Data fork size 0 if there is none
87	4		Resource fork size 0 if there is none
91	4		Creation date and time **Probably a MacOS based timestamp see HFS**
95	4		Modification date and time **Probably a MacOS based timestamp see HFS**
As of MacBinary II, in previous version these values would be all zeroed			
99	2		Get info size 0 if there is none
101	1		Finder flags bits 0-7 Bits 8-15 are defined at offset 73 See section: Finder flags
As of MacBinary III, in previous version these values would be all zeroed			
102	4	"mBIN"	Signature
106	1		Filename script (fdscript of fxinfo)
107	1		Extended finder flags (dfxflags of fxinfo)
108	8	0x00	**Unknown (Unused)**
As of MacBinary II, in previous version these values would be all zeroed			
116	4		Size unpacked

Offset	Size	Value	Description
120	2		Secondary header size 0 if there is none
122	1		MacBinary II version (0x81) MacBinary III version (0x82)
123	1	0x81	Minimum required MacBinary II (or later) version
124	2		CRC-16 of the previous 124 bytes **What CRC algorithm?**
As of MacBinary			
126	2		**Unknown (Reserved for computer type and OS identifier)**

14.2. MacOS finder flags

Value	Identifier	Description
0x0001		is on desk
0x000e		color (bits 1-3)
0x0010		color reserved
0x0020		requires switch launch
0x0040		is shared
0x0080		has no inits
0x0100		inited
0x0200		changed
0x0400		budy
0x0800		**Unknown (bozo?)**
0x1000		system
0x2000		bundle
0x4000		invisible
0x8000		locked

15. Corruption scenarios

The scenarios described below do not seem to be normal behavior.

15.1. Encrypted PFF with encryption type none

Although encryption type is none some PFF files still contain compressible encrypted data. This was found in multiple times in PST files created by Microsoft Exchange Mailbox Merge Program (ExMerge). One of which created by ExMerge v6.5.7529.0.

One of these PST files also did not contain a message store and name-to-identifier-map.

Neither Outlook (MAPI) or scanpst was able to handle these files. Libpff is handles this scenario automatically.

In (at least) one of these files the entries of some data arrays all are encrypted except for the last one. Libpff is able to detect some of these unencrypted data array entries.

15.2. Missing root index node and allocation table pages

Encountered a 1.4 GiB PST file created by ExMerge (unknown version) with zero root index node pages and missing allocation table pages.

Neither Outlook (MAPI) or scanpst was able to handle these files. Libpff is able to handle this scenario only using recovery.

15.3. Split attached message

Encountered a PST file with an e-mail which has 4 e-mail messages attached. These 4 e-mail messages contained parts of a single message. The data of the single message was stored in the message body of the 4 attached e-mails. These 4 attached e-mails claim to have an attachment themselves, but Outlook is not able to access these attachments; neither do they contain any data.

Scanpst did not remark this as a problem. Libpff will not recombine the split attached message, and will tell you it cannot export the attachment of the attached e-mails.

15.4. Extended ASCII strings with Unicode codepage

In a PST file, with invalid encryption type, created by ExMerge v6.5.7529.0,from a ExChange 2003 server with BlackBerry RIM the codepage 1200 was found. Codepage 1200 is not defined by MAPI but used by Windows to indicate Unicode string. So the PST contains (extended) ASCII strings with the Unicode-'codepage'.

For now only UTF-8 strings have been found, but in the Extensible Storage Engine (ESE) Database File (EDB) format codepage 1200 is either used for UTF-8 or UTF-16 little-endian strings.

Libpff as of version 20100429 will handle this scenario automatically.

15.5. Unicode string which contains 16-bit extended ASCII string

In a Unicode PST file, probably copied from an extended ASCII PST file, some messages contain values which are stored as Unicode strings but actually contains an extended ASCII string stored as 16-bit characters.

Libpff does not handle this scenario yet.

15.6. Name-to-id map entry string size that is out of bounds

In a PST file, the string size of a name-to-id map entry is larger than the total name-to-id map strings data. Since the name-to-id map strings are stored without an end-of-string character it is not directly possible to tell the size of the string.

Libpff as of version 20130731 will detect this scenario and mark the name-to-id entry as corrupted.

An alternative approach could be to determine the first valid name-to-id entry string offset after that of the corrupted entry.

15.7. 7c or ac table value array number out of bounds

In a PST file in a 7c or ac table the value array number of a record entry is larger than the number of entries the value array holds.

Libpff as of version 20130804 will detect this scenario and ignore the corrupted record entry and mark the corresponding table as missing record entry data.

16. Notes

TODO: add info about the free map and free page map (header)

16.1. Root items

16.1.1. PST

```
33 message store
97 name-to-id map

290 root folder
301 sub folders (7c table)
302 sub messages (7c table)
303 sub associated items (7c table)

481 empty descriptor
513 Active searches list (see below)
609 Search criteria list (see below)
641 Search gatherer queue (empty descriptor)
673 Search gatherer descriptor (see below)
801 empty descriptor
1549 empty 7c table (template of some kind?)
1550 empty 7c table (template of some kind?)
1551 empty 7c table (template of some kind?)
1552 empty 7c table (template of some kind?)
1579 7c table
```

```
1612 empty 7c table (template?)
1649 empty 7c table (also used by attachments sub item) (template?)
1682 empty 7c table (also used by recipients sub item) (template?)
1718 empty 7c table (also used by unknowns sub item) (template?)
1751 empty 7c table (template?)
1784 7c table (template?)

1840 emails/items table (ac table)
3073 guid lookup table (9c table)

> 8194 related folder items

8742 empty descriptor (Search update queue)
8743 bc table (Search update queue)
8752 empty 7c table (Search contents table)

> 32813 other kind of folders ?

32813 7c table (Sub folders)
32814 empty 7c table (Sub messages)
32815 empty 7c table (Sub associated contents)

32845 7c table (Sub folders)
32846 empty 7c table (Sub messages)
32847 empty 7c table (Sub associated contents)

32877 empty 7c table (Sub folders)
32878 empty 7c table (Sub messages)
32879 empty 7c table (Sub associated contents)

32909 empty 7c table
32910 emails/items table (7c table)
32911 empty 7c table

> 524326 special folders ?

524326 empty descriptor (Search update queue)
524327 bc table (Search update queue)
524336 empty 7c table (Search contents table)

> 1048648 messages ?
```

Active searches list (513)

```
00000000: 23 22 00 00 23 00 08 00 43 00 08 00 63 00 08 00 #"..#... C...c...
00000010: 83 00 08 00 ....

00000000: 23 22 00 00 23 00 08 00 43 00 08 00 63 00 08 00 #"..#... C...c...
00000010: 83 00 08 00
```

List of descriptors?

Search criteria list (609)

```
00000000: 43 21 00 00 63 21 00 00 83 21 00 00 a3 21 00 00 C!..c!.. .!...!..
00000010: c3 21 00 00 23 80 00 00 43 80 00 00 63 80 00 00 .!..#... C...c...
00000020: 83 80 00 00 a3 80 00 00 c3 80 00 00 e3 80 00 00 ........ ........
00000030: 03 81 00 00 23 81 00 00 43 81 00 00 63 81 00 00 ....#... C...c...
00000040: 83 81 00 00 a3 81 00 00........

00000000: 43 21 00 00 63 21 00 00 83 21 00 00 a3 21 00 00 C!..c!.. .!...!..
00000010: c3 21 00 00 23 80 00 00 43 80 00 00 63 80 00 00 .!..#... C...c...
00000020: 83 80 00 00 a3 80 00 00 c3 80 00 00 e3 80 00 00 ........ ........
00000030: 03 81 00 00 23 81 00 00 43 81 00 00 63 81 00 00 ....#... C...c...
00000040: 83 81 00 00 a3 81 00 00........
```

List of descriptors?

Search gatherer descriptor (673)

```
00000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000010: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00 ........ ........
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000050: 00 00 00 00 00 00 00 00........

00000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000010: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00 ........ ........
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........ ........
00000050: 00 00 00 00 00 00 00 00........
```

16.1.2. OST

```
33 message store
97 name-to-id map

290 root folder
```

```
301 sub folders (7c table)
302 sub messages (7c table)
303 sub associated items (7c table)

481 empty descriptor
513 (unknown)
609 (unknown)
641 empty descriptor
673 (unknown)
801 empty descriptor

template tables or used to store empty tables for the specific purpose only once?
1549 empty 7c table (folder related template?)
1550 empty 7c table (message related template?)
1551 empty 7c table (message related template?)
1552 empty 7c table (message related template?)
1556 empty 8c table
1561 empty 7c table (identifier related template?)
1579 7c table with report data (contains IPC, IPM, REPORT.IPM)

1612 empty 7c table (Submit related template?)
1649 empty 7c table (attachments related template?) (attachments sub item)
1682 empty 7c table (recipients related template?) (recipients sub item)
1718 empty 7c table (identifier related template?) (unknowns sub item)
1751 empty 7c table (identifier related template?)
1784 empty 7c table (identifier related template?)

2049 6c table (contains GUID that map to other GUIDs?)
2081 8c table (folder identifier related table? 0x67f4 value related)
2113 7c table (folder identifier releated table? 0x36de value related)
3073 empty 9c table

> 8194 related folder items

8194 folder (no root item)

8205 sub folders
8206 sub messages
8207 sub associated items
8212 empty 8c table
8213 empty 8c table

...

8739 SPAM search folder 2 (Outlook.ItemProcessor) (no root item)

8742 empty descriptor (8739+3)
8743 bc table (contains a single entry 0x660b 0x0003) (8739+4)
8752 sub associated items (empty 7c table)

...
```

```
> 32813 other kind of folders ?

32802 Calendar (no root item)

32813 sub folders
32814 sub messages
32815 sub associated items
32820 empty 8c table
32821 empty 8c table

32825 empty 7c table (32802+23)

...

> 524326 special folders ?

524323 Reminder folder (Outlook.Reminder) (no root item)

524326 empty descriptor
524327 bc table
524336 empty 7c table

524355 To-do search folder (IPF.Task) (no root item)

524358 empty descriptor
524359 bc table (IPM.Appointment)
524368 empty 7c table

524387 ItemProcSearch folder (Outlook.ItemProcessor) (no root item)

524390 empty descriptor
524391 bc table
524400 7c table (messages)

524419 Tracked Mail Processing folder (IPF.Note) (no root item)

524422 empty descriptor
524423 bc table
524432 empty 7c table

> 1048648 messages ?
```

16.1.3. PAB

```
33 message store
39 bc table with "MAPIPDL"
41 0x01 0x00 data type
63 7c table with contacts
95 0x02 0x00 data type
255 empty 7c table
287 0x01 0x00 data type
319 bc table
```

41 (0x29) libpff_table_read: table:

```
00000000: 01 00 00 00 00 00 00 00 00 00 00 00 1e 00 01 30 ........ .......0
00000010: 00 00 00 00 ....
```

95 (0x5f) libpff_table_read: table:

```
00000000: 02 00 00 00 00 00 00 00 00 00 00 00 1e 00 00 e3 ........ ........
00000010: 00 00 00 00 02 01 0b 30 00 00 00 00 .......0 ....
```

287 (0x11f) libpff_table_read: table:

```
00000000: 01 00 00 00 00 00 00 00 00 00 00 00 03 00 00 e4 ........ ........
00000010: 00 00 00 00 ....
```

16.2. GUID identifiers

16.2.1. PST

The record key in the message store item (33) is used as GUID in the entry identifiers for the within the PST file.

```
0x0ff9 (PidTagRecordKey : Record key)
0x0102 (PT_BINARY : Binary data)

GUID : 4a2f9232-b9d7-4afc-a1d5-9634785b50af
```

16.2.2. OST

The 0x6615 value in the message store item (33) is used as GUID in the entry identifiers for the within the OST file.

```
0x6615 (_UNKNOWN_ : Unknown)
0x0048 (PT_CLSID : GUID (128-bit))

GUID : 1b5d8add-509a-4cb8-af41-8ff3e6375fca
```

Note that the entryid object identifier data is different than those used in a PST file.

```
0x36d0 (PidTagIpmAppointmentEntryId : Calendar folder entry identifier)
0x0102 (PT_BINARY : Binary data)

Entry identifier:
Flags: 0x00, 0x00, 0x00, 0x00
Service provider identifier : 77c9cc1c-4915-4c72-bd2f-f28d487b92cc (Unknown)
Object identifier data:
00000000: 01 00

00000000: 8e 5d f2 3a 78 32 7a 41 ae 4c 2c ce 44 e1 ...].:x2 zA.L,.D.
00000010: 8a 88

GUID: 3af25d8e-3278-417a-ae4c-2cce44e18a88

00000010: 00 00 00 19 50 c5

First part of the PRQ_ID_SECURE4 value

00000010:00 00......P. ..


Corresponding item

0x65e2 (PR_CHANGE_KEY : Change key)
0x0102 (PT_BINARY : Binary data)

GUID : af1252d6-dd91-4391-b1fa-ee82341e0c04
Unknown1:
00000000: 00 00 04 12 ....

0x65e3 (PR_PREDECESSOR_CHANGE_LIST : Predecessor change list)
0x0102 (PT_BINARY : Binary data)

Size : 0x16
GUID : 3af25d8e-3278-417a-ae4c-2cce44e18a88
Unknown:
00000000: 00 00 00 19 60 c8 ....`.

Size : 0x14
GUID : af1252d6-dd91-4391-b1fa-ee82341e0c04
Unknown:
00000000: 00 00 04 12
```

```
0x67f4 (PRQ_ID_SECURE4 : )
0x0014 (PT_I8 : Integer 64-bit signed)

integer 64-bit signed : -4228852562310201343 (0xc550190000000001)

6c table

record entry guid: 3af25d8e-3278-417a-ae4c-2cce44e18a88
record entry values array number : 0x0001
record entry value guid : 3af25d8e-3278-417a-ae4c-2cce44e18a88

maps the guid to the last part of the PRQ_ID_SECURE4

8c table

identifier: 0xc550190000000001
descriptor identifier : 0x00008080

maps the PRQ_ID_SECURE4 to a descriptor identifier, requires correction of the lower
four bits
```

But what is record key used for?

```
0x0ff9 (PidTagRecordKey : Record key)
0x0102 (PT_BINARY : Binary data)
GUID : e71f4b30-150f-410d-90f9-1e7d9204d76e
```

16.3. Note

```
Server Failures

0x65e2 (PR_CHANGE_KEY : Change key)
0x0102 (PT_BINARY : Binary data)

GUID : 5003c7c4-af96-4aa3-9d5b-e0c1e039d0ef
Unknown1:
00000000: 00 00 00 12 76 38 ....v8

0x65e3 (PR_PREDECESSOR_CHANGE_LIST : Predecessor change list)
0x0102 (PT_BINARY : Binary data)

Size : 0x16
GUID : 3af25d8e-3278-417a-ae4c-2cce44e18a88
Unknown:
00000000: 00 00 00 3c f4 8e ...<..

Size : 0x16
GUID : 5003c7c4-af96-4aa3-9d5b-e0c1e039d0ef
Unknown:
00000000: 00 00 00 12 76 38 ....v8

0x67f4 (PRQ_ID_SECURE4 : )
0x0014 (PT_I8 : Integer 64-bit signed)

integer 64-bit signed : 929777818772963329 (0xce73c0000000001)
```

```
Inbox

0x65e2 (PR_CHANGE_KEY : Change key)
0x0102 (PT_BINARY : Binary data)

GUID : af1252d6-dd91-4391-b1fa-ee82341e0c04
Unknown1:
00000000: 00 00 04 0f

0x65e3 (PR_PREDECESSOR_CHANGE_LIST : Predecessor change list)
0x0102 (PT_BINARY : Binary data)

Size : 0x16
GUID : 5003c7c4-af96-4aa3-9d5b-e0c1e039d0ef
Unknown:
00000000: 00 00 00 0c 9c d8

Size : 0x14
GUID : af1252d6-dd91-4391-b1fa-ee82341e0c04
Unknown:
00000000: 00 00 04 0f

0x67f4 (PRQ_ID_SECURE4 : )
0x0014 (PT_I8 : Integer 64-bit signed)

integer 64-bit signed : -2209002423085694974 (0xe1580c0000000002)

0xd89c0c0000000002
```

```
Archive Search

0x65e2 (PR_CHANGE_KEY : Change key)
0x0102 (PT_BINARY : Binary data)

GUID : af1252d6-dd91-4391-b1fa-ee82341e0c04
Unknown1:
00000000: 00 00 04 1c ....

0x65e3 (PR_PREDECESSOR_CHANGE_LIST : Predecessor change list)
0x0102 (PT_BINARY : Binary data)

Size : 0x16
GUID : 5003c7c4-af96-4aa3-9d5b-e0c1e039d0ef
Unknown:
00000000: 00 00 00 0c a4 e4 ......

Size : 0x14
GUID : af1252d6-dd91-4391-b1fa-ee82341e0c04
Unknown:
00000000: 00 00 04 1c

0x67f4 (PRQ_ID_SECURE4 : )
0x0014 (PT_I8 : Integer 64-bit signed)

integer 64-bit signed : -2136944829047767038 (0xe2580c0000000002)
```

```
6c table

record entry guid: 5003c7c4-af96-4aa3-9d5b-e0c1e039d0ef
record entry values array number : 0x0002
record entry value guid : 5003c7c4-af96-4aa3-9d5b-e0c1e039d0ef
```

Appendix A: References

[SMITH02]

Title:	**outlook.pst — format of MS Outlook .pst file**
Author(s):	David Smith, Joe Nahmias, Brad Hards, Carl Byington
URL:	http://hg.file-ten-sg.com/libpst/

[LIBESEDB]

Title:	**Extensible Storage Engine (ESE) Database File (EDB) format**
Author(s):	Joachim Metz

Title:	**Extensible Storage Engine (ESE) Database File (EDB) format**
URL:	https://github.com/libyal/libesedb/blob/master/documentation/Extensible%20Storage%20Engine%20(ESE)%20Database%20File%20(EDB)%20format.asciidoc

[LIBFMAPI]

Title:	**Message API (MAPI) definitions**
Auhtor:	Joachim Metz
URL:	https://github.com/libyal/libfmapi/blob/master/documentation/MAPI%20definitions.pdf

[MSDN]

Title:	**Microsoft Developer Network**
URL:	http://msdn.microsoft.com/

[MS-PST]

Title:	**[MS-PST] Outlook Personal Folders File Format (.pst) structure specification**
URL:	http://msdn.microsoft.com/

[MACBINARY]

Title:	**Macintosh Binary Transfer Format ("MacBinary") Standard Proposal**

Title:	**Macintosh Binary Transfer Format ("MacBinary II") Standard Proposal**

Title:	**Macintosh Binary Transfer Format ("MacBinary III") Standard Proposal**

[OPENCHANGE]

Title:	**Openchange MAPI library**
URL:	http://www.openchange.org/index.php

[RFC1950]

Title:	**ZLIB Compressed Data Format Specification**
Version:	3.3
Author(s):	P. Deutsch, J-L. Gailly
Date:	May 1996
URL:	http://www.ietf.org/rfc/rfc1950.txt

[RFC1951]

Title:	DEFLATE Compressed Data Format Specification
Version:	1.3
Author(s):	P. Deutsch
Date:	May 1996
URL:	http://www.ietf.org/rfc/rfc1951.txt

[ROTHMAN99]

Title:	The Compressed RTF Format
Author(s):	Amichai Rothman
URL:	http://www.freeutils.net/source/jtnef/rtfcompressed.jsp

Appendix B: GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the

conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
4. Preserve all the copyright notices of the Document.
5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
8. Include an unaltered copy of this License.
9. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated

in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.