

---

# libpst Utilities - Version 0.6.21

## Packages

This is a fork of the libpst project at SourceForge. Another fork is located at <http://alioth.debian.org/projects/libpst/>

The various source and binary packages are available at <http://www.five-ten-sg.com/libpst/packages/>. The most recent documentation is available at <http://www.five-ten-sg.com/libpst/>.

A Mercurial [<http://www.selenic.com/mercurial/wiki/>] source code repository for this project is available at <http://hg.five-ten-sg.com/libpst/>.

This version can now convert both 32 bit Outlook files (pre 2003), and the 64 bit Outlook 2003 pst files. Utilities are supplied to convert email messages to both mbox and MH mailbox formats, and to DII load file format for use with many of the CT Summation [<http://www.ctsummation.com>] products. Contacts can be converted to a simple list, to vcard format, or to ldif format for import to an LDAP server.

---

# Name

readpst -- convert PST (MS Outlook Personal Folders) files to mbox and other formats

readpst

## Synopsis

readpst [-C] [-D] [-M] [-S] [-V] [-b] [-c *format*] [-d *debug-file*] [-h] [-k] [-o *output-directory*] [-q] [-r] [-w] pstfile

## Description

**readpst** is a program that can read an Outlook PST (Personal Folders) file and convert it into an mbox file, a format suitable for KMail, a recursive mbox structure, or separate emails.

## Options

-C

Decrypt the entire pst file and dump it to stdout.

-D

Include deleted items in the output.

-M

Output messages in MH format as separate files. This will create folders as named in the PST file, and will put each email together with any attachments into its own file. These files will be numbered from 1 to n with no leading zeros.

-S

Output messages into separate files. This will create folders as named in the PST file, and will put each email in its own file. These files will be numbered from 1 increasing in intervals of 1 (ie 1, 2, 3, ...). Any attachments are saved alongside each email as XXXXXXXXXX-attach1, XXXXXXXXXX-attach2 and so on, or with the name of the attachment if one is present.

-V

Show program version and exit.

-b

Do not save the attachments for the RTF format of the email body.

-c *format*

Set the Contact output mode. Use -cv for vcard format or -cl for an email list.

-d *debug-file*

Specify name of debug log file. The log file is not an ascii file, it is a binary file readable by **readpstlog**.

-h

Show summary of options and exit.

-k

Changes the output format to KMail.

**-o *output-directory***

Specifies the output directory. The directory must already exist, and is entered after the PST file is opened, but before any processing of files commences.

**-q**

Changes to silent mode. No feedback is printed to the screen, except for error messages.

**-r**

Changes the output format to Recursive. This will create folders as named in the PST file, and will put all emails in a file called "mbox" inside each folder. These files are then compatible with all mbox-compatible email clients.

**-w**

Overwrite any previous output files. Beware: When used with the -S switch, this will remove all files from the target folder before writing. This is to keep the count of emails and attachments correct.

## See Also

readpstlog(1)

## Author

This manual page was originally written by Dave Smith <dave.s@earthcorp.com>, and updated by Joe Nahmias <joe@nahmias.net> for the Debian GNU/Linux system (but may be used by others). It was subsequently updated by Brad Hards <bradh@frogmouth.net>, and converted to xml format by Carl Byington <carl@five-ten-sg.com>.

## Copyright

Copyright (C) 2002 by David Smith <dave.s@earthcorp.com>. XML version Copyright (C) 2008 by 510 Software Group <carl@five-ten-sg.com>.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

You should have received a copy of the GNU General Public License along with this program; see the file COPYING. If not, please write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

## Version

0.6.21

---

# Name

lspst -- list PST (MS Outlook Personal Folders) file data

lspst

## Synopsis

lspst [-V] [-d *debug-file*] [-h] pstfile

## Options

-V

Show program version and exit.

-d *debug-file*

Specify name of debug log file. The log file is not an ascii file, it is a binary file readable by **readpstlog**.

-h

Show summary of options and exit.

## Description

**lspst** is a program that can read an Outlook PST (Personal Folders) file and produce a simple listing of the data (contacts, email subjects, etc).

## See Also

readpstlog(1)

## Author

lspst was written by Joe Nahmias <joe@nahmias.net> based on readpst. This man page was written by 510 Software Group <carl@five-ten-sg.com>.

## Copyright

Copyright (C) 2004 by Joe Nahmias <joe@nahmias.net>.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

You should have received a copy of the GNU General Public License along with this program; see the file COPYING. If not, please write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

## Version

0.6.21

---

# Name

readpstlog -- convert a **readpst** logfile to text format

readpstlog

## Synopsis

readpstlog [-f *format*] [-t *include-types*] [-x *exclude-types*] logfile

## Description

**readpstlog** is a program that converts the binary logfile generated by **readpst** to a more desirable text format.

## Options

-f *format*

Sets the format of the text log output. Currently, the only valid output formats are T, for single line text, D for the default default multi line format, and I for an indented style with single line text.

-t *include-types*

Print only the specified types of log messages. Types are specified in a comma-delimited list (e.g. 3,10,5,6).

-x *exclude-types*

Exclude the specified types of log messages. Types are specified in a comma-delimited list (e.g. 3,10,5,6).

## Message Types

**readpstlog** understands the following types of log messages:

- 1 File accesses
- 2 Index accesses
- 3 New email found
- 4 Warnings
- 5 Read accesses
- 6 Informational messages
- 7 Main function calls

- 8      Decrypting calls
- 9      Function entries
- 10     Function exits
- 11     HexDump calls

## Author

This manual page was written by Joe Nahmias <joe@nahmias.net> for the Debian GNU/Linux system (but may be used by others). It was converted to xml format by Carl Byington <carl@five-ten-sg.com>.

## TODO

The binary debug log file is generally much larger than the .pst file, so we need to switch to ftello/fseeko there also to handle files larger than 2GB.

## Copyright

Copyright (C) 2002 by David Smith <dave.s@earthcorp.com>. XML version Copyright (C) 2008 by 510 Software Group <carl@five-ten-sg.com>.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

You should have received a copy of the GNU General Public License along with this program; see the file COPYING. If not, please write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

## Version

0.6.21

---

# Name

pst2ldif -- extract contacts from a MS Outlook .pst file in .ldif format

pst2ldif

## Synopsis

pst2ldif [-V] [-b *ldap-base*] [-c *class*] [-C *character-set*] [-d *debug-file*] [-l *extra-line*] [-o] [-h] pstfilename

## Options

-V

Show program version. Subsequent options are then ignored.

-b *ldap-base*

Sets the ldap base value used in the dn records. You probably want to use something like "o=organization, c=US".

-c *class*

Sets the objectClass values for the contact items. This class needs to be defined in the schema used by your LDAP server, and at a minimum it must contain the ldap attributes given below. This option may be specified multiple times to generate entries with multiple object classes.

-C *character-set*

Specify the name of the character set used in your pst file for contacts.

-d *debug-file*

Specify name of debug log file. The log file is not an ascii file, it is a binary file readable by **readpstlog**.

-l *extra-line*

Specify an extra line to be added to each ldap entry. This option may be specified multiple times to add multiple lines to each ldap entry.

-o

Use the old ldap schema, rather than the default new ldap schema. The old schema generates multiple postalAddress attributes for a single entry. The new schema generates a single postalAddress (and homePostalAddress when available) attribute with \$ delimiters as specified in RFC4517. Using the old schema also generates two extra leading entries, one for "dn:ldap base", and one for "dn: cn=root, ldap base".

-h

Show summary of options. Subsequent options are then ignored.

## Description

**pst2ldif** reads the contact information from a MS Outlook .pst file and produces a .ldif file that may be used to import those contacts into an LDAP database. The following ldap attributes are generated for the old ldap schema:

cn	cn
givenName	givenName
sn	sn
personalTitle	title
company	o
mail	mail
postalAddress	postalAddress
l	homePostalAddress
st	l
postalCode	st
c	postalCode
homePhone	c
telephoneNumber	homePhone
	telephoneNumber
	facsimileTelephoneNumber
facsimileTelephoneNumber	
	mobile
mobile	description
description	labeledURI

The following attributes are generated for the new ldap schema:

## Copyright

Copyright (C) 2008 by 510 Software Group <carl@five-ten-sg.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

You should have received a copy of the GNU General Public License along with this program; see the file COPYING. If not, please write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

## Version

0.6.21



---

# Name

pst2dii -- extract email messages from a MS Outlook .pst file in DII load format

pst2dii

## Synopsis

pst2dii [-B *bates-prefix*] [-O *dii-output-file*] [-V] [-b *bates-number*] [-c *bates-color*] [-d *debug-file*] [-f *ttf-font-file*] [-h] [-o *output-directory*] pstfilename

## Options

-B *bates-prefix*

Sets the bates prefix string. The bates sequence number is appended to this string, and printed on each page.

-O *dii-output-file*

Name of the output DII load file.

-V

Show program version. Subsequent options are then ignored.

-b *bates-number*

Starting bates sequence number. The default is zero.

-c *bates-color*

Font color for the bates stamp on each page, specified as 6 hex digits as rrggbb values. The default is ff0000 for bright red.

-d *debug-file*

Specify name of debug log file. The log file is not an ascii file, it is a binary file readable by **readpstlog**.

-f *ttf-font-file*

Specify name of a true type font file. This should be a fixed pitch font.

-h

Show summary of options. Subsequent options are then ignored.

-o *output-directory*

Specifies the output directory. The directory must already exist.

## Description

**pst2dii** reads the email messages from a MS Outlook .pst file and produces a DII load file that may be used to import message summaries into a Summation DII system. The DII output file contains references to the image and attachment files in the output directory.

## Copyright

Copyright (C) 2008 by 510 Software Group <carl@five-ten-sg.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

You should have received a copy of the GNU General Public License along with this program; see the file COPYING. If not, please write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

## Version

0.6.21

---

# Name

outlook.pst -- format of MS Outlook .pst file

outlook.pst

## Synopsis

outlook.pst

## Overview

Each item in a .pst file is identified by two id values ID1 and ID2. There are two separate b-trees indexed by these ID1 and ID2 values. Starting with Outlook 2003, the file format changed from one with 32 bit pointers, to one with 64 bit pointers. We describe both formats here.

## 32 bit File Header

The 32 bit file header is located at offset 0 in the .pst file.

```
0000  21 42 44 4e 49 f8 64 d9 53 4d 0e 00 13 00 01 01
0010  00 00 00 00 00 00 00 00 50 d6 03 00 bd 1e 02 00
0020  08 4c 00 00 00 04 00 00 00 04 00 00 0f 04 00 00
0030  0d 40 00 00 99 0a 01 00 18 04 00 00 0d 40 00 00
0040  0d 40 00 00 11 80 00 00 02 04 00 00 0a 04 00 00
0050  00 04 00 00 00 04 00 00 0f 04 00 00 0f 04 00 00
0060  0f 04 00 00 0d 40 00 00 00 04 00 00 00 04 00 00
0070  04 40 00 00 00 04 00 00 00 04 00 00 00 04 00 00
0080  00 04 00 00 00 04 00 00 00 04 00 00 00 04 00 00
0090  00 04 00 00 00 04 00 00 00 04 00 00 00 04 00 00
00a0  0c 09 00 00 00 00 00 00 00 04 27 00 00 24 23 00
00b0  c0 09 0a 00 00 c8 00 00 bc 1e 02 00 00 7e 0c 00
00c0  b4 1e 02 00 00 54 00 00 01 00 00 00 23 55 44 d1
00d0  5a 4f ce 6b 80 ff ff ff 00 00 00 00 00 00 00 00
00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140  00 00 00 00 00 00 00 00 00 00 00 00 3f ff ff ff
0150  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0160  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0170  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0180  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0190  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01a0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01b0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01c0  ff ff ff ff ff ff ff ff ff ff ff ff ff 80 01 00 00
01d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```

0000 signature      [4 bytes] 0x4e444221 constant
000a indexType     [1 byte]  0x0e      constant
01cd encryptionType [1 byte]  0x01      in this case
00a8 total file size [4 bytes] 0x270400 in this case
00c0 backPointer1   [4 bytes] 0x021eb4 in this case
00c4 offsetIndex1   [4 bytes] 0x005400 in this case
00b8 backPointer2   [4 bytes] 0x021ebc in this case
00bc offsetIndex2   [4 bytes] 0x0c7e00 in this case

```

We only support index types 0x0e and 0x17, and encryption types 0x00, 0x01 and 0x02. Index type 0x0e is the older 32 bit Outlook format. Index type 0x17 is the newer 64 bit Outlook format. Encryption type 0x00 is no encryption, type 0x01 is "compressible" encryption which is a simple substitution cipher, and type 0x02 is "strong" encryption, which seems to be related to a three rotor Enigma cipher.

offsetIndex1 is the file offset of the root of the index1 b-tree, which contains (ID1, offset, size, unknown) tuples for each item in the file. backPointer1 is the value that should appear in the parent pointer of that root node.

offsetIndex2 is the file offset of the root of the index2 b-tree, which contains (ID2, DESC-ID1, LIST-ID1, PARENT-ID2) tuples for each item in the file. backPointer2 is the value that should appear in the parent pointer of that root node.

## 64 bit File Header

The 64 bit file header is located at offset 0 in the .pst file.

```

0000 21 42 44 4e 03 02 23 b2 53 4d 17 00 13 00 01 01
0010 00 00 00 00 00 00 00 00 04 00 00 00 01 00 00 00
0020 8b 00 00 00 00 00 00 00 1d 00 00 00 00 04 00 00
0030 00 04 00 00 04 04 00 00 00 40 00 00 02 00 01 00
0040 00 04 00 00 00 04 00 00 00 04 00 00 00 80 00 00
0050 00 04 00 00 00 04 00 00 00 04 00 00 00 04 00 00
0060 04 04 00 00 04 04 00 00 04 04 00 00 00 04 00 00
0070 00 04 00 00 00 04 00 00 00 04 00 00 00 04 00 00
0080 00 04 00 00 00 04 00 00 00 04 00 00 00 04 00 00
0090 00 04 00 00 00 04 00 00 00 04 00 00 00 04 00 00
00a0 00 04 00 00 00 04 00 00 02 04 00 00 00 00 00 00
00b0 00 00 00 00 00 00 00 00 00 24 04 00 00 00 00 00
00c0 00 44 00 00 00 00 00 00 00 71 03 00 00 00 00 00
00d0 00 22 00 00 00 00 00 00 83 00 00 00 00 00 00 00
00e0 00 6a 00 00 00 00 00 00 8a 00 00 00 00 00 00 00
00f0 00 60 00 00 00 00 00 00 01 00 00 00 00 00 00 00
0100 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180 7f ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0190 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```

```
01a0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01b0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01c0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01d0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01e0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
01f0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0200 80 00 00 00 e8 00 00 00 00 00 00 00 c4 68 cb 89
```

```
0000 signature      [4 bytes] 0x4e444221 constant
000a indexType      [1 byte]  0x17      constant
0201 encryptionType [1 byte]  0x00      in this case
00b8 total file size [8 bytes] 0x042400 in this case
00e8 backPointer1    [8 bytes] 0x00008a in this case
00f0 offsetIndex1    [8 bytes] 0x006000 in this case
00d8 backPointer2    [8 bytes] 0x000083 in this case
00e0 offsetIndex2    [8 bytes] 0x006a00 in this case
```

## 32 bit Index 1 Node

The 32 bit index1 b-tree nodes are 512 byte blocks with the following format.

```
0000 04 00 00 00 8a 1e 02 00 00 1c 0b 00
000c 58 27 03 00 b3 1e 02 00 00 52 00 00
0018 00 00 00 00 00 00 00 00 00 00 00 00
0024 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00
003c 00 00 00 00 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00
006c 00 00 00 00 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00 00 00 00 00
0084 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00
009c 00 00 00 00 00 00 00 00 00 00 00 00
00a8 00 00 00 00 00 00 00 00 00 00 00 00
00b4 00 00 00 00 00 00 00 00 00 00 00 00
00c0 00 00 00 00 00 00 00 00 00 00 00 00
00cc 00 00 00 00 00 00 00 00 00 00 00 00
00d8 00 00 00 00 00 00 00 00 00 00 00 00
00e4 00 00 00 00 00 00 00 00 00 00 00 00
00f0 00 00 00 00 00 00 00 00 00 00 00 00
00fc 00 00 00 00 00 00 00 00 00 00 00 00
0108 00 00 00 00 00 00 00 00 00 00 00 00
0114 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00
012c 00 00 00 00 00 00 00 00 00 00 00 00
0138 00 00 00 00 00 00 00 00 00 00 00 00
0144 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00
015c 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0168 00 00 00 00 00 00 00 00 00 00 00 00
0174 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00
018c 00 00 00 00 00 00 00 00 00 00 00 00
0198 00 00 00 00 00 00 00 00 00 00 00 00
01a4 00 00 00 00 00 00 00 00 00 00 00 00
01b0 00 00 00 00 00 00 00 00 00 00 00 00
01bc 00 00 00 00 00 00 00 00 00 00 00 00
01c8 00 00 00 00 00 00 00 00 00 00 00 00
01d4 00 00 00 00 00 00 00 00 00 00 00 00
01e0 00 00 00 00 00 00 00 00 00 00 00 00
01ec 00 00 00 00 02 29 0c 02 80 80 b6 4a
01f8 b4 1e 02 00 27 9c cc 56
```

```
01f0 itemCount      [1 byte] 0x02      in this case
01f1 maxItemCount   [1 byte] 0x29      constant
01f2 itemSize       [1 byte] 0x0c      constant
01f3 nodeLevel      [1 byte] 0x02      in this case
01f8 backPointer    [4 bytes] 0x021eb4 in this case
```

The itemCount specifies the number of 12 byte records that are active. The nodeLevel is non-zero for this style of nodes. The leaf nodes have a different format. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a triple of (ID1, backPointer, offset) where the offset points to the next deeper node in the tree, the backPointer value must match the backPointer in that deeper node, and ID1 is the lowest ID1 value in the subtree.

## 64 bit Index 1 Node

The 64 bit index1 b-tree nodes are 512 byte blocks with the following format.

```
0000 04 00 00 00 00 00 00 00 88 00 00 00
000C 00 00 00 00 00 00 48 00 00 00 00 00
0018 74 00 00 00 00 00 00 00 00 00 00 00
0024 00 00 00 00 00 00 54 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00
003C 00 00 00 00 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00
006C 00 00 00 00 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00 00 00 00 00
0084 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00
009C 00 00 00 00 00 00 00 00 00 00 00 00
00A8 00 00 00 00 00 00 00 00 00 00 00 00
00B4 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00
00CC 00 00 00 00 00 00 00 00 00 00 00 00
00D8 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00E4 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00
00FC 00 00 00 00 00 00 00 00 00 00 00
0108 00 00 00 00 00 00 00 00 00 00 00
0114 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00
012C 00 00 00 00 00 00 00 00 00 00 00
0138 00 00 00 00 00 00 00 00 00 00 00
0144 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00
015C 00 00 00 00 00 00 00 00 00 00 00
0168 00 00 00 00 00 00 00 00 00 00 00
0174 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00
018C 00 00 00 00 00 00 00 00 00 00 00
0198 00 00 00 00 00 00 00 00 00 00 00
01A4 00 00 00 00 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00
01BC 00 00 00 00 00 00 00 00 00 00 00
01C8 00 00 00 00 00 00 00 00 00 00 00
01D4 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 02 14 18 01
01EC 00 00 00 00 80 80 8a 60 68 e5 b5 19
01F8 8a 00 00 00 00 00 00 00 00
```

```
01e8 itemCount      [1 byte] 0x02      in this case
01e9 maxItemCount   [1 byte] 0x14      constant
01ea itemSize       [1 byte] 0x18      constant
01eb nodeLevel      [1 byte] 0x01      in this case
01f8 backPointer    [8 bytes] 0x00008a in this case
```

The itemCount specifies the number of 24 byte records that are active. The nodeLevel is non-zero for this style of nodes. The leaf nodes have a different format. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a triple of (ID1, backPointer, offset) where the offset points to the next deeper node in the tree, the backPointer value must match the backPointer in that deeper node, and ID1 is the lowest ID1 value in the subtree.

## 32 bit Index 1 Leaf Node

The 32 bit index1 b-tree leaf nodes are 512 byte blocks with the following format.

```
0000 04 00 00 00 00 58 00 00 64 00 0f 00
000c 08 00 00 00 80 58 00 00 ac 00 06 00
0018 0c 00 00 00 40 59 00 00 ac 00 06 00
0024 10 00 00 00 00 5a 00 00 bc 00 03 00
0030 14 00 00 00 00 5b 00 00 a4 00 02 00
003c 18 00 00 00 c0 5b 00 00 64 00 02 00
0048 1c 00 00 00 40 5c 00 00 5c 00 02 00
0054 50 00 00 00 80 62 00 00 60 00 02 00
```

```
0060 74 00 00 00 00 77 00 00 5e 00 02 00
006c 7c 00 00 00 80 77 00 00 66 00 02 00
0078 84 00 00 00 00 76 00 00 ca 00 02 00
0084 88 00 00 00 00 63 00 00 52 00 02 00
0090 90 00 00 00 00 79 00 00 58 00 02 00
009c cc 00 00 00 c0 61 00 00 76 00 02 00
00a8 e0 00 00 00 00 61 00 00 74 00 02 00
00b4 f4 00 00 00 80 65 00 00 6e 00 02 00
00c0 8c 01 00 00 40 60 00 00 70 00 02 00
00cc ea 01 00 00 80 61 00 00 10 00 02 00
00d8 ec 01 00 00 40 8a 00 00 f3 01 02 00
00e4 f0 01 00 00 80 93 00 00 f4 1f 02 00
00f0 fa 01 00 00 c0 7f 00 00 10 00 02 00
00fc 00 02 00 00 00 89 00 00 34 01 02 00
0108 1c 02 00 00 40 ec 00 00 12 06 02 00
0114 22 02 00 00 00 84 00 00 10 00 02 00
0120 24 02 00 00 c0 ea 00 00 3c 01 02 00
012c 40 02 00 00 00 f4 00 00 0a 06 02 00
0138 46 02 00 00 40 8c 00 00 10 00 02 00
0144 48 02 00 00 80 f2 00 00 36 01 02 00
0150 64 02 00 00 80 fb 00 00 bf 07 02 00
015c 6a 02 00 00 80 63 00 00 10 00 02 00
0168 6c 02 00 00 40 fa 00 00 2a 01 02 00
0174 6c 02 00 00 40 fa 00 00 2a 01 02 00
0180 6c 02 00 00 40 fa 00 00 2a 01 02 00
018c 6c 02 00 00 40 fa 00 00 2a 01 02 00
0198 6c 02 00 00 40 fa 00 00 2a 01 02 00
01a4 6c 02 00 00 40 fa 00 00 2a 01 02 00
01b0 64 02 00 00 80 fb 00 00 bf 07 02 00
01bc 64 02 00 00 80 fb 00 00 bf 07 02 00
01c8 64 02 00 00 80 fb 00 00 bf 07 02 00
01d4 64 02 00 00 80 fb 00 00 bf 07 02 00
01e0 64 02 00 00 80 fb 00 00 bf 07 02 00
01ec 00 00 00 00 1f 29 0c 00 80 80 5b b3
01f8 5a 67 01 00 4f ae 70 a7
```

```
01f0 itemCount      [1 byte] 0x1f      in this case
01f1 maxItemCount   [1 byte] 0x29      constant
01f2 itemSize       [1 byte] 0x0c      constant
01f3 nodeLevel      [1 byte] 0x00      defines a leaf node
01f8 backPointer    [4 bytes] 0x01675a in this case
```

The itemCount specifies the number of 12 byte records that are active. The nodeLevel is zero for these leaf nodes. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a tuple of (ID1, offset, size, unknown) The two low order bits of the ID1 value seem to be flags. I have never seen a case with bit zero set. Bit one indicates that the item is *not* encrypted. Note that references to these ID1 values elsewhere may have the low order bit set (and I don't know what that means), but when we do the search in this tree we need to clear that bit so that we can find the correct item.

## 64 bit Index 1 Leaf Node

---



The 64 bit index1 b-tree leaf nodes are 512 byte blocks with the following format.

```
0000 04 00 00 00 00 00 00 00 00 58 00 00
000C 00 00 00 00 6c 00 05 00 00 00 00 00
0018 08 00 00 00 00 00 00 00 80 58 00 00
0024 00 00 00 00 b4 00 06 00 d8 22 37 08
0030 0c 00 00 00 00 00 00 00 80 59 00 00
003C 00 00 00 00 ac 00 07 00 d8 22 37 08
0048 10 00 00 00 00 00 00 00 40 5a 00 00
0054 00 00 00 00 bc 00 03 00 d8 22 37 08
0060 14 00 00 00 00 00 00 00 40 5b 00 00
006C 00 00 00 00 a4 00 02 00 d8 22 37 08
0078 18 00 00 00 00 00 00 00 00 5c 00 00
0084 00 00 00 00 64 00 02 00 d8 22 37 08
0090 1c 00 00 00 00 00 00 00 80 5c 00 00
009C 00 00 00 00 5c 00 02 00 d8 22 37 08
00A8 24 00 00 00 00 00 00 00 80 5d 00 00
00B4 00 00 00 00 72 00 02 00 d8 22 37 08
00C0 34 00 00 00 00 00 00 00 00 70 00 00
00CC 00 00 00 00 8c 00 02 00 00 0d 00 00
00D8 38 00 00 00 00 00 00 00 c0 71 00 00
00E4 00 00 00 00 5c 00 02 00 d8 22 9c 00
00F0 40 00 00 00 00 00 00 00 40 72 00 00
00FC 00 00 00 00 26 00 02 00 d8 22 9c 00
0108 4c 00 00 00 00 00 00 00 80 5f 00 00
0114 00 00 00 00 3e 00 02 00 d8 22 9c 00
0120 5c 00 00 00 00 00 00 00 c0 76 00 00
012C 00 00 00 00 8c 00 02 00 d8 22 9c 00
0138 64 00 00 00 00 00 00 00 40 75 00 00
0144 00 00 00 00 76 00 02 00 d8 22 9c 00
0150 6c 00 00 00 00 00 00 00 c0 73 00 00
015C 00 00 00 00 5e 00 02 00 d8 22 9c 00
0168 70 00 00 00 00 00 00 00 80 72 00 00
0174 00 00 00 00 1e 01 02 00 d8 22 9c 00
0180 70 00 00 00 00 00 00 00 80 72 00 00
018C 00 00 00 00 1e 01 02 00 d8 22 9c 00
0198 70 00 00 00 00 00 00 00 80 72 00 00
01A4 00 00 00 00 1e 01 02 00 d8 22 9c 00
01B0 74 00 00 00 00 00 00 00 40 74 00 00
01BC 00 00 00 00 e0 00 02 00 d8 22 9c 00
01C8 7c 00 00 00 00 00 00 00 80 77 00 00
01D4 00 00 00 00 dc 00 02 00 d8 22 9c 00
01E0 00 00 00 00 00 00 00 00 10 14 18 00
01EC 00 00 00 00 80 80 88 48 3f 50 0b 04
01F8 88 00 00 00 00 00 00 00
```

```
01e8 itemCount      [1 byte] 0x10      in this case
01e9 maxItemCount   [1 byte] 0x14      constant
01ea itemSize       [1 byte] 0x18      constant
01eb nodeLevel      [1 byte] 0x00      defines a leaf node
01f8 backPointer    [8 bytes] 0x000088 in this case
```

The itemCount specifies the number of 24 byte records that are active. The nodeLevel is zero for these leaf nodes. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a tuple of (ID1, offset, size, unknown) The two low order bits of the ID1 value seem to be flags. I have never seen a case with bit zero set. Bit one indicates that the item is *not* encrypted. Note that references to these ID1 values elsewhere may have the low order bit set (and I don't know what that means), but when we do the search in this tree we need to clear that bit so that we can find the correct item.

## 32 bit Index 2 Node

The 32 bit index2 b-tree nodes are 512 byte blocks with the following format.

```
0000 21 00 00 00 bb 1e 02 00 00 e2 0b 00
000c 64 78 20 00 8c 1e 02 00 00 dc 0b 00
0018 00 00 00 00 00 00 00 00 00 00 00 00
0024 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00
003c 00 00 00 00 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00
006c 00 00 00 00 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00 00 00 00 00
0084 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00
009c 00 00 00 00 00 00 00 00 00 00 00 00
00a8 00 00 00 00 00 00 00 00 00 00 00 00
00b4 00 00 00 00 00 00 00 00 00 00 00 00
00c0 00 00 00 00 00 00 00 00 00 00 00 00
00cc 00 00 00 00 00 00 00 00 00 00 00 00
00d8 00 00 00 00 00 00 00 00 00 00 00 00
00e4 00 00 00 00 00 00 00 00 00 00 00 00
00f0 00 00 00 00 00 00 00 00 00 00 00 00
00fc 00 00 00 00 00 00 00 00 00 00 00 00
0108 00 00 00 00 00 00 00 00 00 00 00 00
0114 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00
012c 00 00 00 00 00 00 00 00 00 00 00 00
0138 00 00 00 00 00 00 00 00 00 00 00 00
0144 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00
015c 00 00 00 00 00 00 00 00 00 00 00 00
0168 00 00 00 00 00 00 00 00 00 00 00 00
0174 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00
018c 00 00 00 00 00 00 00 00 00 00 00 00
0198 00 00 00 00 00 00 00 00 00 00 00 00
01a4 00 00 00 00 00 00 00 00 00 00 00 00
01b0 00 00 00 00 00 00 00 00 00 00 00 00
01bc 00 00 00 00 00 00 00 00 00 00 00 00
01c8 00 00 00 00 00 00 00 00 00 00 00 00
01d4 00 00 00 00 00 00 00 00 00 00 00 00
01e0 00 00 00 00 00 00 00 00 00 00 00 00
```

```
01ec 00 00 00 00 02 29 0c 02 81 81 b2 60
01f8 bc 1e 02 00 7e 70 dc e3
```

```
01f0 itemCount      [1 byte] 0x02      in this case
01f1 maxItemCount   [1 byte] 0x29      constant
01f2 itemSize       [1 byte] 0x0c      constant
01f3 nodeLevel      [1 byte] 0x02      in this case
01f8 backPointer    [4 bytes] 0x021ebc in this case
```

The itemCount specifies the number of 12 byte records that are active. The nodeLevel is non-zero for this style of nodes. The leaf nodes have a different format. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a triple of (ID2, backPointer, offset) where the offset points to the next deeper node in the tree, the backPointer value must match the backPointer in that deeper node, and ID2 is the lowest ID2 value in the subtree.

## 64 bit Index 2 Node

The 64 bit index2 b-tree nodes are 512 byte blocks with the following format.

```
0000 21 00 00 00 00 00 00 00 00 77 00 00 00
000C 00 00 00 00 00 00 56 00 00 00 00 00 00
0018 4c 06 00 00 00 00 00 00 00 82 00 00 00
0024 00 00 00 00 00 00 68 00 00 00 00 00 00
0030 4f 80 00 00 00 00 00 00 00 84 00 00 00
003C 00 00 00 00 00 00 6e 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00
006C 00 00 00 00 00 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00 00 00 00 00 00
0084 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00
009C 00 00 00 00 00 00 00 00 00 00 00 00 00
00A8 00 00 00 00 00 00 00 00 00 00 00 00 00
00B4 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00
00CC 00 00 00 00 00 00 00 00 00 00 00 00 00
00D8 00 00 00 00 00 00 00 00 00 00 00 00 00
00E4 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00
00FC 00 00 00 00 00 00 00 00 00 00 00 00 00
0108 00 00 00 00 00 00 00 00 00 00 00 00 00
0114 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00
012C 00 00 00 00 00 00 00 00 00 00 00 00 00
0138 00 00 00 00 00 00 00 00 00 00 00 00 00
0144 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00
015C 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0168 00 00 00 00 00 00 00 00 00 00 00 00
0174 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00
018C 00 00 00 00 00 00 00 00 00 00 00 00
0198 00 00 00 00 00 00 00 00 00 00 00 00
01A4 00 00 00 00 00 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00 00
01BC 00 00 00 00 00 00 00 00 00 00 00 00
01C8 00 00 00 00 00 00 00 00 00 00 00 00
01D4 00 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 03 14 18 01
01EC 00 00 00 00 81 81 83 6a 49 da f3 d3
01F8 83 00 00 00 00 00 00 00 00 00 00 00
```

```
01e8 itemCount      [1 byte] 0x03      in this case
01e9 maxItemCount   [1 byte] 0x14      constant
01ea itemSize       [1 byte] 0x18      constant
01eb nodeLevel      [1 byte] 0x01      in this case
01f8 backPointer    [8 bytes] 0x000083 in this case
```

The itemCount specifies the number of 24 byte records that are active. The nodeLevel is non-zero for this style of nodes. The leaf nodes have a different format. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a triple of (ID2, backPointer, offset) where the offset points to the next deeper node in the tree, the backPointer value must match the backPointer in that deeper node, and ID2 is the lowest ID2 value in the subtree.

## 32 bit Index 2 Leaf Node

The 32 bit index2 b-tree leaf nodes are 512 byte blocks with the following format.

```
0000 21 00 00 00 38 e6 00 00 00 00 00 00 00 00 00 00
0010 61 00 00 00 2c a8 02 00 36 a8 02 00 00 00 00 00
0020 22 01 00 00 20 a2 02 00 00 00 00 00 22 01 00 00
0030 2d 01 00 00 88 7b 03 00 00 00 00 00 00 00 00 00
0040 2e 01 00 00 08 00 00 00 00 00 00 00 00 00 00 00
0050 2f 01 00 00 0c 00 00 00 00 00 00 00 00 00 00 00
0060 e1 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 01 02 00 00 b4 e4 02 00 00 00 00 00 00 00 00 00
0080 61 02 00 00 a0 e4 02 00 00 00 00 00 00 00 00 00
0090 0d 06 00 00 04 00 00 00 00 00 00 00 00 00 00 00
00A0 0e 06 00 00 08 00 00 00 00 00 00 00 00 00 00 00
00B0 0f 06 00 00 0c 00 00 00 00 00 00 00 00 00 00 00
00C0 10 06 00 00 10 00 00 00 00 00 00 00 00 00 00 00
00D0 2b 06 00 00 84 00 00 00 00 00 00 00 00 00 00 00
00E0 4c 06 00 00 1c 00 00 00 00 00 00 00 00 00 00 00
00F0 71 06 00 00 18 00 00 00 00 00 00 00 00 00 00 00
0100 92 06 00 00 14 00 00 00 00 00 00 00 00 00 00 00
0110 23 22 00 00 14 a0 02 00 00 00 00 00 22 01 00 00
0120 26 22 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```

0130 27 22 00 00 1c a0 02 00 00 00 00 00 00 00 00 00 00 00
0140 22 80 00 00 50 00 00 00 00 00 00 00 22 01 00 00
0150 2d 80 00 00 f8 9f 02 00 00 00 00 00 00 00 00 00
0160 2e 80 00 00 08 00 00 00 00 00 00 00 00 00 00 00
0170 2f 80 00 00 34 e6 00 00 00 00 00 00 00 00 00 00
0180 42 80 00 00 3c 6d 02 00 00 00 00 00 22 80 00 00
0190 4d 80 00 00 04 00 00 00 00 00 00 00 00 00 00 00
01A0 4e 80 00 00 10 6d 02 00 00 00 00 00 00 00 00 00
01B0 4f 80 00 00 ec 23 00 00 00 00 00 00 00 00 00 00
01C0 62 80 00 00 38 78 02 00 00 00 00 00 22 01 00 00
01D0 6d 80 00 00 34 78 02 00 00 00 00 00 00 00 00 00
01E0 6e 80 00 00 08 00 00 00 00 00 00 00 00 00 00 00
01F0 10 1f 10 00 81 81 a0 9a ae 1e 02 00 89 44 6a 0f

```

```

01f0 itemCount      [1 byte] 0x10      in this case
01f1 maxItemCount   [1 byte] 0x1f      constant
01f2 itemSize       [1 byte] 0x10      constant
01f3 nodeLevel      [1 byte] 0x00      in this case
01f8 backPointer    [4 bytes] 0x021eae in this case

```

The itemCount specifies the number of 16 byte records that are active. The nodeLevel is zero for these leaf nodes. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a tuple of (ID2, DESC-ID1, LIST-ID1, PARENT-ID2)

## 64 bit Index 2 Leaf Node

The 64 bit index2 b-tree leaf nodes are 512 byte blocks with the following format.

```

0000 21 00 00 00 00 00 00 00 74 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
0020 61 00 00 00 00 00 00 00 34 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
0040 22 01 00 00 00 00 00 00 4c 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 22 01 00 00 02 00 00 00
0060 2d 01 00 00 00 00 00 00 70 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
0080 2e 01 00 00 00 00 00 00 08 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
00A0 2f 01 00 00 00 00 00 00 0c 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
00C0 e1 01 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 d8 e3 13 00
00E0 01 02 00 00 00 00 00 00 8c 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 b0 e3 13 00
0100 61 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 d8 e3 13 00
0120 0d 06 00 00 00 00 00 00 04 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
0140 0e 06 00 00 00 00 00 00 08 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00

```

```
0160 0f 06 00 00 00 00 00 00 0c 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
0180 10 06 00 00 00 00 00 00 10 00 00 00 00 00 00 00
0190 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
01A0 2b 06 00 00 00 00 00 00 24 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
01C0 71 06 00 00 00 00 00 00 18 00 00 00 00 00 00 00
01D0 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00
01E0 00 00 00 00 00 00 00 00 0e 0f 20 00 00 00 00 00
01F0 81 81 77 56 f8 32 43 49 77 00 00 00 00 00 00 00
```

```
01e8 itemCount      [1 byte] 0x0e      in this case
01e9 maxItemCount   [1 byte] 0x0f      constant
01ea itemSize       [1 byte] 0x20      constant
01eb nodeLevel      [1 byte] 0x00      defines a leaf node
01f8 backPointer    [8 bytes] 0x000077 in this case
```

The itemCount specifies the number of 32 byte records that are active. The nodeLevel is zero for these leaf nodes. The backPointer must match the backPointer from the triple that pointed to this node.

Each item in this node is a tuple of (ID2, DESC-ID1, LIST-ID1, PARENT-ID2)

## 32 bit Associated List Item 0x0002

Contains associations between id1 and id2 for the items controlled by the record. In the above 32 bit leaf node, we have a tuple of (0x61, 0x02a82c, 0x02a836, 0) 0x02a836 is the ID1 of the associated list, and we can lookup that ID1 value in the index1 b-tree to find the (offset,size) of the data in the .pst file.

```
0000 02 00 01 00 9f 81 00 00 30 a8 02 00 00 00 00 00
```

```
0000 signature      [2 bytes] 0x0002      constant
0002 count          [2 bytes] 0x0001      in this case
      repeating
0004 id2            [4 bytes] 0x00819f      in this case
0008 id             [4 bytes] 0x02a830      in this case
000c table2         [4 bytes] 0           in this case
```

## 64 bit Associated List Item 0x0002

Contains associations between id1 and id2 for the items controlled by the record.

```
0000 02 00 02 00 00 00 00 00 92 06 00 00 00 00 00 00
0010 a8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 3f 80 00 00 00 00 00 00 98 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00
```

```
0000 signature      [2 bytes] 0x0002      constant
0002 count          [2 bytes] 0x0002      in this case
```

0004	unknown	[4 bytes]	0	possibly constant
	repeating			
0008	id2	[4 bytes]	0x000692	in this case
000c	unknown1	[2 bytes]	0	may be a count or size
000e	unknown2	[2 bytes]	0	may be a count or size
0010	id	[8 bytes]	0x0000a8	in this case
0018	table2	[8 bytes]	0	in this case

## Associated Descriptor Item 0xbcec

Contains information about the item, which may be email, contact, or other outlook types. In the above leaf node, we have a tuple of (0x21, 0x00e638, 0, 0) 0x00e638 is the ID1 of the associated descriptor, and we can lookup that ID1 value in the index1 b-tree to find the (offset,size) of the data in the .pst file.

```
0000 3c 01 ec bc 20 00 00 00 00 00 00 00 b5 02 06 00
0010 40 00 00 00 f9 0f 02 01 60 00 00 00 01 30 1e 00
0020 80 00 00 00 04 30 1e 00 00 00 00 00 df 35 03 00
0030 ff 00 00 00 e0 35 02 01 a0 00 00 00 e2 35 02 01
0040 e0 00 00 00 e3 35 02 01 c0 00 00 00 e4 35 02 01
0050 00 01 00 00 e5 35 02 01 20 01 00 00 e6 35 02 01
0060 40 01 00 00 e7 35 02 01 60 01 00 00 1e 66 0b 00
0070 00 00 00 00 ff 67 03 00 00 00 00 00 d2 7f 17 d8
0080 64 8c d5 11 83 24 00 50 04 86 95 45 53 74 61 6e
0090 6c 65 79 00 00 00 00 d2 7f 17 d8 64 8c d5 11 83
00A0 24 00 50 04 86 95 45 22 80 00 00 00 00 00 00 d2
00B0 7f 17 d8 64 8c d5 11 83 24 00 50 04 86 95 45 42
00C0 80 00 00 00 00 00 00 d2 7f 17 d8 64 8c d5 11 83
00D0 24 00 50 04 86 95 45 a2 80 00 00 00 00 00 00 d2
00E0 7f 17 d8 64 8c d5 11 83 24 00 50 04 86 95 45 c2
00F0 80 00 00 00 00 00 00 d2 7f 17 d8 64 8c d5 11 83
0100 24 00 50 04 86 95 45 e2 80 00 00 00 00 00 00 d2
0110 7f 17 d8 64 8c d5 11 83 24 00 50 04 86 95 45 02
0120 81 00 00 00 00 00 00 d2 7f 17 d8 64 8c d5 11 83
0130 24 00 50 04 86 95 45 62 80 00 00 00 0b 00 00 00
0140 0c 00 14 00 7c 00 8c 00 93 00 ab 00 c3 00 db 00
0150 f3 00 0b 01 23 01 3b 01
```

0000	indexOffset	[2 bytes]	0x013c	in this case
0002	signature	[2 bytes]	0xbcec	constant
0004	b5offset	[4 bytes]	0x0020	index reference

Note the signature of 0xbcec. There are other descriptor block formats with other signatures. Note the indexOffset of 0x013c - starting at that position in the descriptor block, we have an array of two byte integers. The first integer (0x000b) is a (count-1) of the number of overlapping pairs following the count. The first pair is (0, 0xc), the next pair is (0xc, 0x14) and the last (12th) pair is (0x123, 0x13b). These pairs are (start,end+1) offsets of items in this block. So we have count+2 integers following the count value.

Note the b5offset of 0x0020, which is a type that I will call an index reference. Such index references have at least two different forms, and may point to data either in this block, or in some other block. External pointer references have the low order 4 bits all set, and are ID2 values that can be used to fetch data. This value of 0x0020 is an internal

pointer reference, which needs to be right shifted by 4 bits to become 0x0002, which is then a byte offset to be added to the above indexOffset plus two (to skip the count), so it points to the (0xc, 0x14) pair.

So far we have only described internal index references where the high order 16 bits are zero. That suffices for single descriptor blocks. But in the case of the type 0x0101 descriptor block, we have an array of subblocks. In this case, the high order 16 bits of an internal index reference are used to select the subblock. Each subblock starts with a 16 bit indexOffset which points to the count and array of 16 bit integer pairs which are offsets in the current subblock.

Finally, we have the offset and size of the "b5" block located at offset 0xc with a size of 8 bytes in this descriptor block. The "b5" block has the following format:

0000	signature	[2 bytes]	0x02b5	constant
0002	datasize	[2 bytes]	0x0006	constant +2 for 8 byte entries
0004	descoffset	[4 bytes]	0x0040	index reference

Note the descoffset of 0x0040, which again is an index reference. In this case, it is an internal pointer reference, which needs to be right shifted by 4 bits to become 0x0004, which is then a byte offset to be added to the above indexOffset plus two (to skip the count), so it points to the (0x14, 0x7c) pair. The datasize (6) plus the b5 code (02) gives the size of the entries, in this case 8 bytes. We now have the offset 0x14 of the descriptor array, composed of 8 byte entries. Each descriptor entry has the following format:

0000	itemType	[2 bytes]
0002	referenceType	[2 bytes]
0004	value	[4 bytes]

For some reference types (2, 3, 0xb) the value is used directly. Otherwise, the value is an index reference, which is either an ID2 value, or an offset, to be right shifted by 4 bits and used to fetch a pair from the index table to find the offset and size of the item in this descriptor block.

The following reference types are known, but not all of these are implemented in the code yet.

0x0002	- Signed 16bit value
0x0003	- Signed 32bit value
0x0004	- 4-byte floating point
0x0005	- Floating point double
0x0006	- Signed 64-bit int
0x0007	- Application Time
0x000A	- 32-bit error value
0x000B	- Boolean (non-zero = true)
0x000D	- Embedded Object
0x0014	- 8-byte signed integer (64-bit)
0x001E	- Null terminated String
0x001F	- Unicode string
0x0040	- Systeime - Filetime structure
0x0048	- OLE Guid
0x0102	- Binary data
0x1003	- Array of 32bit values
0x1014	- Array of 64bit values
0x101E	- Array of Strings



0x1102 - Array of Binary data

The following item types are known, but not all of these are implemented in the code yet.

0002 Alternate recipient allowed  
0003 Extended Attributes Table  
0017 Importance Level  
001a IPM Context, message class  
0023 Global delivery report requested  
0026 Priority  
0029 Read Receipt  
002b Reassignment Prohibited  
002e Original Sensitivity  
0036 Sensitivity  
0037 Email Subject  
0039 Client submit time / date sent  
003b Outlook Address of Sender  
003f Outlook structure describing the recipient  
0040 Name of the Outlook recipient structure  
0041 Outlook structure describing the sender  
0042 Name of the Outlook sender structure  
0043 Another structure describing the recipient  
0044 Name of the second recipient structure  
004f Reply-To Outlook Structure  
0050 Name of the Reply-To structure  
0051 Outlook Name of recipient  
0052 Second Outlook name of recipient  
0057 My address in TO field  
0058 My address in CC field  
0059 Message addressed to me  
0063 Response requested  
0064 Sender's Address access method (SMTP, EX)  
0065 Sender's Address  
0070 Conversation topic, processed subject (with Fwd:, Re, ... removed)  
0071 Conversation index  
0072 Original display BCC  
0073 Original display CC  
0074 Original display TO  
0075 Recipient Address Access Method (SMTP, EX)  
0076 Recipient's Address  
0077 Second Recipient Access Method (SMTP, EX)  
0078 Second Recipient Address  
007d Email Header. This is the header that was attached to the email  
0c17 Reply Requested  
0c19 Second sender structure  
0c1a Name of second sender structure  
0c1d Second outlook name of sender  
0c1e Second sender access method (SMTP, EX)  
0c1f Second Sender Address  
0e01 Delete after submit  
0e02 BCC Addresses

0e03 CC Addresses  
 0e04 SentTo Address  
 0e06 Date.  
 0e07 Flag bits  
     0x01 - Read  
     0x02 - Unmodified  
     0x04 - Submit  
     0x08 - Unsent  
     0x10 - Has Attachments  
     0x20 - From Me  
     0x40 - Associated  
     0x80 - Resend  
     0x100 - RN Pending  
     0x200 - NRN Pending  
 0e08 Message Size  
 0e0a Sentmail EntryID  
 0elf Compressed RTF in Sync  
 0e20 Attachment Size  
 0ff9 binary record header  
 1000 Plain Text Email Body. Does not exist if the email doesn't have a plain<sub>↵</sub>  
 text version  
 1006 RTF Sync Body CRC  
 1007 RTF Sync Body character count  
 1008 RTF Sync body tag  
 1009 RTF Compressed body  
 1010 RTF whitespace prefix count  
 1011 RTF whitespace tailing count  
 1013 HTML Email Body. Does not exist if the email doesn't have an HTML version  
 1035 Message ID  
 1042 In-Reply-To or Parent's Message ID  
 1046 Return Path  
 3001 Folder Name? I have seen this value used for the contacts record aswell  
 3002 Address Type  
 3003 Contact Address  
 3004 Comment  
 3007 Date item creation  
 3008 Date item modification  
 300b binary record header  
 35df Valid Folder Mask  
 35e0 binary record contains a reference to "Top of Personal Folder" item  
 35e2 binary record contains a reference to default outbox item  
 35e3 binary record contains a reference to "Deleted Items" item  
 35e4 binary record contains a reference to sent items folder item  
 35e5 binary record contains a reference to user views folder item  
 35e6 binary record contains a reference to common views folder item  
 35e7 binary record contains a reference to "Search Root" item  
 3602 the number of emails stored in a folder  
 3603 the number of unread emails in a folder  
 360a Has Subfolders  
 3613 the folder content description  
 3617 Associate Content count  
 3701 Binary Data attachment  
 3704 Attachment Filename  
 3705 Attachement method

3707 Attachment Filename long  
 370b Attachment Position  
 370e Attachment mime encoding  
 3710 Attachment mime Sequence  
 3a00 Contact's Account name  
 3a01 Contact Alternate Recipient  
 3a02 Callback telephone number  
 3a03 Message Conversion Prohibited  
 3a05 Contacts Suffix  
 3a06 Contacts First Name  
 3a07 Contacts Government ID Number  
 3a08 Business Telephone Number  
 3a09 Home Telephone Number  
 3a0a Contacts Initials  
 3a0b Keyword  
 3a0c Contact's Language  
 3a0d Contact's Location  
 3a0e Mail Permission  
 3a0f MHS Common Name  
 3a10 Organizational ID #  
 3a11 Contacts Surname  
 3a12 original entry id  
 3a13 original display name  
 3a14 original search key  
 3a15 Default Postal Address  
 3a16 Company Name  
 3a17 Job Title  
 3a18 Department Name  
 3a19 Office Location  
 3a1a Primary Telephone  
 3a1b Business Phone Number 2  
 3a1c Mobile Phone Number  
 3a1d Radio Phone Number  
 3a1e Car Phone Number  
 3a1f Other Phone Number  
 3a20 Transmittable Display Name  
 3a21 Pager Phone Number  
 3a22 user certificate  
 3a23 Primary Fax Number  
 3a24 Business Fax Number  
 3a25 Home Fax Number  
 3a26 Business Address Country  
 3a27 Business Address City  
 3a28 Business Address State  
 3a29 Business Address Street  
 3a2a Business Postal Code  
 3a2b Business PO Box  
 3a2c Telex Number  
 3a2d ISDN Number  
 3a2e Assistant Phone Number  
 3a2f Home Phone 2  
 3a30 Assistant's Name  
 3a40 Can receive Rich Text  
 3a41 Wedding Anniversary

3a42 Birthday  
 3a43 Hobbies  
 3a44 Middle Name  
 3a45 Display Name Prefix (Title)  
 3a46 Profession  
 3a47 Preferred By Name  
 3a48 Spouse's Name  
 3a49 Computer Network Name  
 3a4a Customer ID  
 3a4b TTY/TDD Phone  
 3a4c Ftp Site  
 3a4d Gender  
 3a4e Manager's Name  
 3a4f Nickname  
 3a50 Personal Home Page  
 3a51 Business Home Page  
 3a57 Company Main Phone  
 3a58 childrens names  
 3a59 Home Address City  
 3a5a Home Address Country  
 3a5b Home Address Postal Code  
 3a5c Home Address State or Province  
 3a5d Home Address Street  
 3a5e Home Address Post Office Box  
 3a5f Other Address City  
 3a60 Other Address Country  
 3a61 Other Address Postal Code  
 3a62 Other Address State  
 3a63 Other Address Street  
 3a64 Other Address Post Office box  
 65e3 Entry ID  
 67f2 Attachment ID2 value  
 67ff Password checksum  
 6f02 Secure HTML Body  
 6f04 Secure Text Body  
 7c07 Top of folders RecID  
 8005 Contact Fullname  
 801a Home Address  
 801b Business Address  
 801c Other Address  
 8045 Work Address Street  
 8046 Work Address City  
 8047 Work Address State  
 8048 Work Address Postal Code  
 8049 Work Address Country  
 804a Work Address Post Office Box  
 8082 Email Address 1 Transport  
 8083 Email Address 1 Address  
 8084 Email Address 1 Description  
 8085 Email Address 1 Record  
 8092 Email Address 2 Transport  
 8093 Email Address 2 Address  
 8094 Email Address 2 Description  
 8095 Email Address 2 Record

80a2 Email Address 3 Transport  
80a3 Email Address 3 Address  
80a4 Email Address 3 Description  
80a5 Email Address 3 Record  
80d8 Internet Free/Busy  
8205 Appointment shows as  
8208 Appointment Location  
820d Appointment start  
820e Appointment end  
8214 Label for appointment  
8215 All day appointment flag  
8231 Recurrence type  
8232 Recurrence description  
8234 TimeZone of times  
8235 Recurrence Start Time  
8236 Recurrence End Time  
8501 Reminder minutes before appointment start  
8503 Reminder alarm  
8516 Common Time Start  
8517 Common Time End  
851f Play reminder sound filename  
8530 Followup String  
8534 Mileage  
8535 Billing Information  
8554 Outlook Version  
8560 Appointment Reminder Time  
8700 Journal Entry Type  
8706 Start Timestamp  
8708 End Timestamp  
8712 Journal Entry Type - duplicate?

## Associated Descriptor Item 0x7cec

This style of descriptor block is similar to the 0xbcec format.

```
0000 7a 01 ec 7c 40 00 00 00 00 00 00 00 b5 04 02 00
0010 60 00 00 00 7c 18 60 00 60 00 62 00 65 00 20 00
0020 00 00 80 00 00 00 00 00 00 00 03 00 20 0e 0c 00
0030 04 03 1e 00 01 30 2c 00 04 0b 1e 00 03 37 28 00
0040 04 0a 1e 00 04 37 14 00 04 05 03 00 05 37 10 00
0050 04 04 1e 00 07 37 24 00 04 09 1e 00 08 37 20 00
0060 04 08 02 01 0a 37 18 00 04 06 03 00 0b 37 08 00
0070 04 02 1e 00 0d 37 1c 00 04 07 1e 00 0e 37 40 00
0080 04 10 02 01 0f 37 30 00 04 0c 1e 00 11 37 34 00
0090 04 0d 1e 00 12 37 3c 00 04 0f 1e 00 13 37 38 00
00A0 04 0e 03 00 f2 67 00 00 04 00 03 00 f3 67 04 00
00B0 04 01 03 00 09 69 44 00 04 11 03 00 fa 7f 5c 00
00C0 04 15 40 00 fb 7f 4c 00 08 13 40 00 fc 7f 54 00
00D0 08 14 03 00 fd 7f 48 00 04 12 0b 00 fe 7f 60 00
00E0 01 16 0b 00 ff 7f 61 00 01 17 45 82 00 00 00 00
00F0 45 82 00 00 78 3c 00 00 ff ff ff ff 49 1e 00 00
```

```

0100 06 00 00 00 00 00 00 00 a0 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 c0 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 40 dd a3
0140 57 45 b3 0c 00 40 dd a3 57 45 b3 0c 02 00 00 00
0150 00 00 fa 10 3e 2a 86 48 86 f7 14 03 0a 03 02 01
0160 4a 2e 20 44 61 76 69 64 20 4b 61 72 61 6d 27 73
0170 20 42 69 72 74 68 64 61 79 00 06 00 00 00 0c 00
0180 14 00 ea 00 f0 00 55 01 60 01 79 01

```

```

0000 indexOffset    [2 bytes] 0x017a    in this case
0002 signature     [2 bytes] 0x7cec    constant
0004 7coffset      [4 bytes] 0x0040    index reference

```

Note the signature of 0x7cec. There are other descriptor block formats with other signatures. Note the indexOffset of 0x017a - starting at that position in the descriptor block, we have an array of two byte integers. The first integer (0x0006) is a (count-1) of the number of overlapping pairs following the count. The first pair is (0, 0xc), the next pair is (0xc, 0x14) and the last (7th) pair is (0x160, 0x179). These pairs are (start,end+1) offsets of items in this block. So we have count+2 integers following the count value.

Note the 7coffset of 0x0040, which is an index reference. In this case, it is an internal reference pointer, which needs to be right shifted by 4 bits to become 0x0004, which is then a byte offset to be added to the above indexOffset plus two (to skip the count), so it points to the (0x14, 0xea) pair. We have the offset and size of the "7c" block located at offset 0x14 with a size of 214 bytes in this case. The "7c" block starts with a header with the following format:

```

0000 signature     [1 bytes] 0x7c      constant
0001 itemCount     [1 bytes] 0x18      in this case
0002 unknown       [2 bytes] 0x0060    in this case
0004 unknown       [2 bytes] 0x0060    in this case
0006 unknown       [2 bytes] 0x0062    in this case
0008 recordSize    [2 bytes] 0x0065    in this case
000a b5Offset      [4 bytes] 0x0020    index reference
000e index2Offset  [4 bytes] 0x0080    index reference
0012 unknown       [2 bytes] 0x0000    in this case
0014 unknown       [2 bytes] 0x0000    in this case

```

Note the b5Offset of 0x0020, which is an index reference. In this case, it is an internal reference pointer, which needs to be right shifted by 4 bits to become 0x0002, which is then a byte offset to be added to the above indexOffset plus two (to skip the count), so it points to the (0xc, 0x14) pair. Finally, we have the offset and size of the "b5" block located at offset 0xc with a size of 8 bytes in this descriptor block. The "b5" block has the following format:

```

0000 signature     [2 bytes] 0x04b5    constant
0002 datasize      [2 bytes] 0x0002    +4 for 6 byte entries in this case
0004 descoffset    [4 bytes] 0x0060    index reference

```

Note the descoffset of 0x0060, which again is an index reference. In this case, it is an internal pointer reference, which needs to be right shifted by 4 bits to become 0x0006, which is then a byte offset to be added to the above indexOffset plus two (to skip the count), so it points to the (0xea, 0xf0) pair. The datasize (2) plus the b5 code (04) gives the size

of the entries, in this case 6 bytes. We now have the offset 0xea of an unused block of data in an unknown format, composed of 6 byte entries. That gives us  $(0xf0 - 0xea)/6 = 1$ , so we have a recordCount of one.

We have seen cases where the descOffset in the b5 block is zero, and the index2Offset in the 7c block is zero. This has been seen for objects that seem to be attachments on messages that have been read. Before the message was read, it did not have any attachments.

Note the index2Offset above of 0x0080, which again is an index reference. In this case, it is an internal pointer reference, which needs to be right shifted by 4 bits to become 0x0008, which is then a byte offset to be added to the above indexOffset plus two (to skip the count), so it points to the (0xf0, 0x155) pair. This is an array of tables of four byte integers. We will call these the IND2 tables. The size of each of these tables is specified by the recordSize field of the "7c" header. The number of these tables is the above recordCount value derived from the "b5" block.

Now the remaining data in the "7c" block after the header starts at offset 0x2a. There should be itemCount 8 byte items here, with the following format:

```
0000  referenceType  [2 bytes]
0002  itemType      [2 bytes]
0004  ind2Offset    [2 bytes]
0006  size          [1 byte]
0007  unknown       [1 byte]
```

The ind2Offset is a byte offset into the current IND2 table of some value. If that is a four byte integer value, then once we fetch that, we have the same triple (item type, reference type, value) as we find in the 0xbcec style descriptor blocks. If not, then this value is used directly. These 8 byte descriptors are processed recordCount times, each time using the next IND2 table. The item and reference types are as described above for the 0xbcec format descriptor block.

## 32 bit Associated Descriptor Item 0x0101

This descriptor block contains a list of ID1 values. It is used when an ID1 (that would normally point to a type 0x7cec or 0xbcec descriptor block) contains more data than can fit in any single descriptor of those types. In this case, it points to a type 0x0101 block, which contains a list of ID1 values that themselves point to the actual descriptor blocks. The total length value in the 0x0101 header is the sum of the lengths of the blocks pointed to by the list of ID1 values. The result is an array of subblocks, that may contain index references where the high order 16 bits specify which descriptor subblock to use. Only the first descriptor subblock contains the signature (0xbcec or 0x7cec).

```
0000  01 01 02 00 26 28 00 00 18 77 0c 00 b8 04 00 00

0000  signature      [2 bytes] 0x0101    constant
0002  count          [2 bytes] 0x0002    in this case
0004  total length   [4 bytes] 0x002826   in this case
      repeating
0008  id1            [4 bytes] 0x0c7718   in this case
000c  id1            [4 bytes] 0x0004b8   in this case
```

## 64 bit Associated Descriptor Item 0x0101

This descriptor block contains a list of ID1 values.

0000 01 01 02 00 ea 29 00 00 10 83 00 00 00 00 00 00  
0010 1c 83 00 00 00 00 00 00

0000 signature [2 bytes] 0x0101 constant  
0002 count [2 bytes] 0x0002 in this case  
0004 total length [4 bytes] 0x0029ea in this case  
repeating  
0008 id1 [8 bytes] 0x008310 in this case  
0010 id1 [8 bytes] 0x00831c in this case