```
**********************************************************
   53990 Mon Sep 16 15:02:46 2013
new/usr/src/cmd/lofiadm/main.c
*** NO COMMENTS ***
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  * Copyright 2012 Joyent, Inc.  All rights reserved.
  25  */

  27 /*
  28  * lofiadm - administer lofi(7d). Very simple, add and remove file<->device
  29  * associations, and display status. All the ioctls are private between
  30  * lofi and lofiadm, and so are very simple - device information is
  31  * communicated via a minor number.
  32  */

  34 #include <sys/types.h>
  35 #include <sys/param.h>
  36 #include <sys/lofi.h>
  37 #include <sys/stat.h>
  38 #include <sys/sysmacros.h>
  39 #include <netinet/in.h>
  40 #include <stdio.h>
  41 #include <fcntl.h>
  42 #include <locale.h>
  43 #include <string.h>
  44 #include <strings.h>
  45 #include <errno.h>
  46 #include <stdlib.h>
  47 #include <unistd.h>
  48 #include <stropts.h>
  49 #include <libdevinfo.h>
  50 #include <libgen.h>
  51 #include <ctype.h>
  52 #include <dlfcn.h>
  53 #include <limits.h>
  54 #include <security/cryptoki.h>
  55 #include <cryptoutil.h>
  56 #include <sys/crypto/ioctl.h>
  57 #include <sys/crypto/ioctladmin.h>
  58 #include "utils.h"
  59 #include <LzmaEnc.h>

  61 /* Only need the IV len #defines out of these files, nothing else. */
```



```
  62 #include <aes/aes_impl.h>
  63 #include <des/des_impl.h>
  64 #include <blowfish/blowfish_impl.h>

  66 static const char USAGE[] =
  67         "Usage: %s [-r] -a file [ device ] "
  67         "Usage: %s -a file [ device ] "
  68         " [-c aes-128-cbc|aes-192-cbc|aes-256-cbc|des3-cbc|blowfish-cbc]"
  69         " [-e] [-k keyfile] [-T [token]:[manuf]:[serial]:key]\n"
  70         "       %s -d file | device\n"
  71         "       %s -C [gzip|gzip-6|gzip-9|lzma] [-s segment_size] file\n"
  72         "       %s -U file\n"
  73         "       %s [ file | device ]\n";

  75 typedef struct token_spec {
  76         char    *name;
  77         char    *mfr;
  78         char    *serno;
  79         char    *key;
  80 } token_spec_t;
_____unchanged_portion_omitted_

 361 /*
 362  * Add a device association. If devicename is NULL, let the driver
 363  * pick a device.
 364  */
 365 static void
 366 add_mapping(int lfd, const char *devicename, const char *filename,
 367     mech_alias_t *cipher, const char *rkey, size_t rksz, boolean_t rdonly)
 367     mech_alias_t *cipher, const char *rkey, size_t rksz)
 368 {
 369         struct lofi_ioctl li;

 371         li.li_readonly = rdonly;

 373         li.li_crypto_enabled = B_FALSE;
 374         if (cipher != NULL) {
 375                 /* set up encryption for mapped file */
 376                 li.li_crypto_enabled = B_TRUE;
 377                 (void) strlcpy(li.li_cipher, cipher->name,
 378                     sizeof (li.li_cipher));
 379                 if (rksz > sizeof (li.li_key)) {
 380                         die(gettext("key too large"));
 381                 }
 382                 bcopy(rkey, li.li_key, rksz);
 383                 li.li_key_len = rksz << 3;      /* convert to bits */

 385                 li.li_iv_type = cipher->iv_type;
 386                 li.li_iv_len = cipher->iv_len;  /* 0 when no iv needed */
 387                 switch (cipher->iv_type) {
 388                 case IVM_ENC_BLKNO:
 389                         (void) strlcpy(li.li_iv_cipher, cipher->iv_name,
 390                             sizeof (li.li_iv_cipher));
 391                         break;
 392                 case IVM_NONE:
 393                         /* FALLTHROUGH */
 394                 default:
 395                         break;
 396                 }
 397         }

 399         if (devicename == NULL) {
 400                 int     minor;

 402                 /* pick one via the driver */
 403                 minor = lofi_map_file(lfd, li, filename);
```

```
404                 /* if mapping succeeds, print the one picked */
405                 (void) printf("/dev/%s/%d\n", LOFI_BLOCK_NAME, minor);
406                 return;
407         }

409         /* use device we were given */
410         li.li_minor = name_to_minor(devicename);
411         if (li.li_minor == 0) {
412                 die(gettext("malformed device name %s\n"), devicename);
413         }
414         (void) strlcpy(li.li_filename, filename, sizeof (li.li_filename));

416         /* if device is already in use li.li_minor won't change */
417         if (ioctl(lfd, LOFI_MAP_FILE_MINOR, &li) == -1) {
418                 if (errno == ENOTSUP)
419                         warn(gettext("encrypting compressed files is "
420                             "unsupported"));
421                 die(gettext("could not map file %s to %s"), filename,
422                     devicename);
423         }
424         wait_until_dev_complete(li.li_minor);
425 }
```

*unchanged_portion_omitted_*

```
492 /*
493  * Print the list of all the mappings, including a header.
494  */
495 static void
496 print_mappings(int fd)
497 {
498         struct lofi_ioctl li;
499         int     minor;
500         int     maxminor;
501         char    path[MAXPATHLEN];
502         char    options[MAXPATHLEN] = { 0 };
500         char    options[MAXPATHLEN];

504         li.li_minor = 0;
505         if (ioctl(fd, LOFI_GET_MAXMINOR, &li) == -1) {
506                 die("ioctl");
507         }
508         maxminor = li.li_minor;

510         (void) printf(FORMAT, gettext("Block Device"), gettext("File"),
511             gettext("Options"));
512         for (minor = 1; minor <= maxminor; minor++) {
513                 li.li_minor = minor;
514                 if (ioctl(fd, LOFI_GET_FILENAME, &li) == -1) {
515                         if (errno == ENXIO)
516                                 continue;
517                         warn("ioctl");
518                         break;
519                 }
520                 (void) snprintf(path, sizeof (path), "/dev/%s/%d",
521                     LOFI_BLOCK_NAME, minor);

523                 options[0] = '\0';

525                 /*
526                  * Encrypted lofi and compressed lofi are mutually exclusive.
527                  */
528                 if (li.li_crypto_enabled)
529                         (void) snprintf(options, sizeof (options),
530                             gettext("Encrypted"));
531                 else if (li.li_algorithm[0] != '\0')
532                         (void) snprintf(options, sizeof (options),
```



```
533                             gettext("Compressed(%s)"), li.li_algorithm);
534                 if (li.li_readonly) {
535                         if (strlen(options) != 0) {
536                                 (void) strlcat(options, ",", sizeof (options));
537                                 (void) strlcat(options, "Readonly",
538                                     sizeof (options));
539                         } else {
540                                 (void) snprintf(options, sizeof (options),
541                                     gettext("Readonly"));
542                         }
543                 }
544                 if (strlen(options) == 0)
529                 else
545                         (void) snprintf(options, sizeof (options), "-");

547                 (void) printf(FORMAT, path, li.li_filename, options);
548         }
549 }
```

*unchanged_portion_omitted_*

```
1790 int
1791 main(int argc, char *argv[])
1792 {
1793         int     lfd;
1794         int     c;
1795         const char *devicename = NULL;
1796         const char *filename = NULL;
1797         const char *algname = COMPRESS_ALGORITHM;
1798         int     openflag;
1799         int     minor;
1800         int     compress_index;
1801         uint32_t segsize = SEGSIZE;
1802         static char *lofictl = "/dev/" LOFI_CTL_NAME;
1803         boolean_t force = B_FALSE;
1804         const char *pname;
1805         boolean_t errflag = B_FALSE;
1806         boolean_t addflag = B_FALSE;
1807         boolean_t rdflag = B_FALSE;
1808         boolean_t deleteflag = B_FALSE;
1809         boolean_t ephflag = B_FALSE;
1810         boolean_t compressflag = B_FALSE;
1811         boolean_t uncompressflag = B_FALSE;
1812         /* the next two work together for -c, -k, -T, -e options only */
1813         boolean_t need_crypto = B_FALSE;        /* if any -c, -k, -T, -e */
1814         boolean_t cipher_only = B_TRUE;         /* if -c only */
1815         const char *keyfile = NULL;
1816         mech_alias_t *cipher = NULL;
1817         token_spec_t *token = NULL;
1818         char    *rkey = NULL;
1819         size_t  rksz = 0;
1820         char realfilename[MAXPATHLEN];

1822         pname = getpname(argv[0]);

1824         (void) setlocale(LC_ALL, "");
1825         (void) textdomain(TEXT_DOMAIN);

1827         while ((c = getopt(argc, argv, "a:c:Cd:efk:o:rs:T:U")) != EOF) {
1811         while ((c = getopt(argc, argv, "a:c:Cd:efk:o:s:T:U")) != EOF) {
1828                 switch (c) {
1829                 case 'a':
1830                         addflag = B_TRUE;
1831                         if ((filename = realpath(optarg, realfilename)) == NULL)
1832                                 die("%s", optarg);
1833                         if (((argc - optind) > 0) && (*argv[optind] != '-')) {
1834                                 /* optional device */
```

```
1835                                devicename = argv[optind];
1836                                optind++;
1837                        }
1838                        break;
1839                case 'C':
1840                        compressflag = B_TRUE;
1841                        if (((argc - optind) > 1) && (*argv[optind] != '-')) {
1842                                /* optional algorithm */
1843                                algname = argv[optind];
1844                                optind++;
1845                        }
1846                        check_algorithm_validity(algname, &compress_index);
1847                        break;
1848                case 'c':
1849                        /* is the chosen cipher allowed? */
1850                        if ((cipher = ciph2mech(optarg)) == NULL) {
1851                                errflag = B_TRUE;
1852                                warn(gettext("cipher %s not allowed\n"),
1853                                    optarg);
1854                        }
1855                        need_crypto = B_TRUE;
1856                        /* cipher_only is already set */
1857                        break;
1858                case 'd':
1859                        deleteflag = B_TRUE;
1860                        minor = name_to_minor(optarg);
1861                        if (minor != 0)
1862                                devicename = optarg;
1863                        else {
1864                                if ((filename = realpath(optarg,
1865                                    realfilename)) == NULL)
1866                                        die("%s", optarg);
1867                        }
1868                        break;
1869                case 'e':
1870                        ephflag = B_TRUE;
1871                        need_crypto = B_TRUE;
1872                        cipher_only = B_FALSE;  /* need to unset cipher_only */
1873                        break;
1874                case 'f':
1875                        force = B_TRUE;
1876                        break;
1877                case 'k':
1878                        keyfile = optarg;
1879                        need_crypto = B_TRUE;
1880                        cipher_only = B_FALSE;  /* need to unset cipher_only */
1881                        break;
1882                case 'r':
1883                        rdflag = B_TRUE;
1884                        break;
1885                case 's':
1886                        segsize = convert_to_num(optarg);
1887                        if (segsize < DEV_BSIZE || !ISP2(segsize))
1888                                die(gettext("segment size %s is invalid "
1889                                    "or not a multiple of minimum block "
1890                                    "size %ld\n"), optarg, DEV_BSIZE);
1891                        break;
1892                case 'T':
1893                        if ((token = parsetoken(optarg)) == NULL) {
1894                                errflag = B_TRUE;
1895                                warn(
1896                                    gettext("invalid token key specifier %s\n"),
1897                                    optarg);
1898                        }
1899                        need_crypto = B_TRUE;
1900                        cipher_only = B_FALSE;  /* need to unset cipher_only */
```



```
1901                        break;
1902                case 'U':
1903                        uncompressflag = B_TRUE;
1904                        break;
1905                case '?':
1906                default:
1907                        errflag = B_TRUE;
1908                        break;
1909                }
1910        }

1912        /* Check for mutually exclusive combinations of options */
1913        if (errflag ||
1914            (addflag && deleteflag) ||
1915            (rdflag && !addflag) ||
1916            (!addflag && need_crypto) ||
1917            ((compressflag || uncompressflag) && (addflag || deleteflag)))
1918                usage(pname);

1920        /* ephemeral key, and key from either file or token are incompatible */
1921        if (ephflag && (keyfile != NULL || token != NULL)) {
1922                die(gettext("ephemeral key cannot be used with keyfile"
1923                    " or token key\n"));
1924        }

1926        /*
1927         * "-c" but no "-k", "-T", "-e", or "-T -k" means derive key from
1928         * command line passphrase
1929         */

1931        switch (argc - optind) {
1932        case 0: /* no more args */
1933                if (compressflag || uncompressflag)     /* needs filename */
1934                        usage(pname);
1935                break;
1936        case 1:
1937                if (addflag || deleteflag)
1938                        usage(pname);
1939                /* one arg means compress/uncompress the file ... */
1940                if (compressflag || uncompressflag) {
1941                        if ((filename = realpath(argv[optind],
1942                            realfilename)) == NULL)
1943                                die("%s", argv[optind]);
1944                /* ... or without options means print the association */
1945                } else {
1946                        minor = name_to_minor(argv[optind]);
1947                        if (minor != 0)
1948                                devicename = argv[optind];
1949                        else {
1950                                if ((filename = realpath(argv[optind],
1951                                    realfilename)) == NULL)
1952                                        die("%s", argv[optind]);
1953                        }
1954                }
1955                break;
1956        default:
1957                usage(pname);
1958                break;
1959        }

1961        if (addflag || compressflag || uncompressflag)
1962                check_file_validity(filename);

1964        if (filename && !valid_abspath(filename))
1965                exit(E_ERROR);
```

```
1967         /*
1968          * Here, we know the arguments are correct, the filename is an
1969          * absolute path, it exists and is a regular file. We don't yet
1970          * know that the device name is ok or not.
1971          */

1973         openflag = O_EXCL;
1974         if (addflag || deleteflag || compressflag || uncompressflag)
1975                 openflag |= O_RDWR;
1976         else
1977                 openflag |= O_RDONLY;
1978         lfd = open(lofictl, openflag);
1979         if (lfd == -1) {
1980                 if ((errno == EPERM) || (errno == EACCES)) {
1981                         die(gettext("you do not have permission to perform "
1982                             "that operation.\n"));
1983                 } else {
1984                         die(gettext("open: %s"), lofictl);
1985                 }
1986                 /*NOTREACHED*/
1987         }

1989         /*
1990          * No passphrase is needed for ephemeral key, or when key is
1991          * in a file and not wrapped by another key from a token.
1992          * However, a passphrase is needed in these cases:
1993          * 1. cipher with no ephemeral key, key file, or token,
1994          *    in which case the passphrase is used to build the key
1995          * 2. token with an optional cipher or optional key file,
1996          *    in which case the passphrase unlocks the token
1997          * If only the cipher is specified, reconfirm the passphrase
1998          * to ensure the user hasn't mis-entered it.  Otherwise, the
1999          * token will enforce the token passphrase.
2000          */
2001         if (need_crypto) {
2002                 CK_SESSION_HANDLE       sess;

2004                 /* pick a cipher if none specified */
2005                 if (cipher == NULL)
2006                         cipher = DEFAULT_CIPHER;

2008                 if (!kernel_cipher_check(cipher))
2009                         die(gettext(
2010                             "use \"cryptoadm list -m\" to find available "
2011                             "mechanisms\n"));

2013                 init_crypto(token, cipher, &sess);

2015                 if (cipher_only) {
2016                         getkeyfromuser(cipher, &rkey, &rksz);
2017                 } else if (token != NULL) {
2018                         getkeyfromtoken(sess, token, keyfile, cipher,
2019                             &rkey, &rksz);
2020                 } else {
2021                         /* this also handles ephemeral keys */
2022                         getkeyfromfile(keyfile, cipher, &rkey, &rksz);
2023                 }

2025                 end_crypto(sess);
2026         }

2028         /*
2029          * Now to the real work.
2030          */
2031         if (addflag)
2032                 add_mapping(lfd, devicename, filename, cipher, rkey, rksz,
```



```
2033                     rdflag);
2012                 add_mapping(lfd, devicename, filename, cipher, rkey, rksz);
2034         else if (compressflag)
2035                 lofi_compress(&lfd, filename, compress_index, segsize);
2036         else if (uncompressflag)
2037                 lofi_uncompress(lfd, filename);
2038         else if (deleteflag)
2039                 delete_mapping(lfd, devicename, filename, force);
2040         else if (filename || devicename)
2041                 print_one_mapping(lfd, devicename, filename);
2042         else
2043                 print_mappings(lfd);

2045         if (lfd != -1)
2046                 (void) close(lfd);
2047         closelib();
2048         return (E_SUCCESS);
2049 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   18081 Mon Sep 16 15:02:47 2013
new/usr/src/man/man1m/lofiadm.1m
*** NO COMMENTS ***
**********************************************************
'\" te
.\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH LOFIADM 1M "Aug 28, 2013"
.TH LOFIADM 1M "Aug 31, 2009"
.SH NAME
lofiadm \- administer files available as block devices through lofi
.SH SYNOPSIS
.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-a\fR \fIfile\fR [\fIdevice\fR]
\fB/usr/sbin/lofiadm\fR \fB-a\fR \fIfile\fR [\fIdevice\fR]
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-a\fR \fIfile\fR [\f
\fB/usr/sbin/lofiadm\fR \fB-c\fR \fIcrypto_algorithm\fR \fB-a\fR \fIfile\fR [\fI
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-k\fR \fIraw_key_fil
\fB/usr/sbin/lofiadm\fR \fB-c\fR \fIcrypto_algorithm\fR \fB-k\fR \fIraw_key_file
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-T\fR \fItoken_key\f
\fB/usr/sbin/lofiadm\fR \fB-c\fR \fIcrypto_algorithm\fR \fB-T\fR \fItoken_key\fR
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-T\fR \fItoken_key\f
\fB/usr/sbin/lofiadm\fR \fB-c\fR \fIcrypto_algorithm\fR \fB-T\fR \fItoken_key\fR
     \fB-k\fR \fIwrapped_key_file\fR \fB-a\fR \fIfile\fR [\fIdevice\fR]
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-e\fR \fB-a\fR \fIfi
\fB/usr/sbin/lofiadm\fR \fB-c\fR \fIcrypto_algorithm\fR \fB-e\fR \fB-a\fR \fIfil
.fi

.LP
.nf
\fBlofiadm\fR \fB-C\fR \fIalgorithm\fR [\fB-s\fR \fIsegment_size\fR] \fIfile\fR
\fB/usr/sbin/lofiadm\fR \fB-C\fR \fIalgorithm\fR [\fB-s\fR \fIsegment_size\fR] \
.fi

.LP
.nf
\fBlofiadm\fR \fB-d\fR \fIfile\fR | \fIdevice\fR
\fB/usr/sbin/lofiadm\fR \fB-d\fR \fIfile\fR | \fIdevice\fR
.fi

.LP
.nf
```



```
\fBlofiadm\fR \fB-U\fR \fIfile\fR
\fB/usr/sbin/lofiadm\fR \fB-U\fR \fIfile\fR
.fi

.LP
.nf
\fBlofiadm\fR [ \fIfile\fR | \fIdevice\fR]
\fB/usr/sbin/lofiadm\fR [ \fIfile\fR | \fIdevice\fR]
.fi

.SH DESCRIPTION
.sp
.LP
\fBlofiadm\fR administers \fBlofi\fR, the loopback file driver. \fBlofi\fR
allows a file to be associated with a block device. That file can then be
accessed through the block device. This is useful when the file contains an
image of some filesystem (such as a floppy or \fBCD-ROM\fR image), because the
block device can then be used with the normal system utilities for mounting,
checking or repairing filesystems. See \fBfsck\fR(1M) and \fBmount\fR(1M).
.sp
.LP
Use \fBlofiadm\fR to add a file as a loopback device, remove such an
association, or print information about the current associations.
.sp
.LP
Encryption and compression options are mutually exclusive on the command line.
Further, an encrypted file cannot be compressed later, nor can a compressed
file be encrypted later.
.sp
.LP
The \fBlofi\fR driver is not available and will not work inside a zone.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-a\fR \fIfile\fR [\fIdevice\fR]\fR
.ad
.sp .6
.RS 4n
Add \fIfile\fR as a block device.
.sp
If \fIdevice\fR is not specified, an available device is picked.
.sp
If \fIdevice\fR is specified, \fBlofiadm\fR attempts to assign it to
\fIfile\fR. \fIdevice\fR must be available or \fBlofiadm\fR will fail. The
ability to specify a device is provided for use in scripts that wish to
reestablish a particular set of associations.
.RE

.sp
.ne 2
.na
\fB\fB-C\fR {\fIgzip\fR | \fIgzip-N\fR | \fIlzma\fR}\fR
.ad
.sp .6
.RS 4n
Compress the file with the specified compression algorithm.
.sp
The \fBgzip\fR compression algorithm uses the same compression as the
open-source \fBgzip\fR command. You can specify the \fBgzip\fR level by using
the value \fBgzip-\fR\fIN\fR where \fIN\fR is 6 (fast) or 9 (best compression
ratio). Currently, \fBgzip\fR, without a number, is equivalent to \fBgzip-6\fR
(which is also the default for the \fBgzip\fR command).
```

```
117 .sp
118 \fIlzma\fR stands for the LZMA (Lempel-Ziv-Markov) compression algorithm.
119 .sp
120 Note that you cannot write to a compressed file, nor can you mount a compressed
121 file read/write.
122 .RE

124 .sp
125 .ne 2
126 .na
127 \fB\fB-d\fR \fIfile\fR | \fIdevice\fR\fR
128 .ad
129 .sp .6
130 .RS 4n
131 Remove an association by \fIfile\fR or \fIdevice\fR name, if the associated
132 block device is not busy, and deallocates the block device.
133 .RE

135 .sp
136 .ne 2
137 .na
138 \fB\fB-r\fR
139 .ad
140 .sp .6
141 .RS 4n
142 If the \fB-r\fR option is specified before the \fB-a\fR option, the
143 \fIdevice\fR will be opened read-only.
144 .RE

146 .sp
147 .ne 2
148 .na
149 \fB\fB-s\fR \fIsegment_size\fR\fR
150 .ad
151 .sp .6
152 .RS 4n
153 The segment size to use to divide the file being compressed. \fIsegment_size\fR
154 can be an integer multiple of 512.
155 .RE

157 .sp
158 .ne 2
159 .na
160 \fB\fB-U\fR \fIfile\fR\fR
161 .ad
162 .sp .6
163 .RS 4n
164 Uncompress a compressed file.
165 .RE

167 .sp
168 .LP
169 The following options are used when the file is encrypted:
170 .sp
171 .ne 2
172 .na
173 \fB\fB-c\fR \fIcrypto_algorithm\fR\fR
174 .ad
175 .sp .6
176 .RS 4n
177 Select the encryption algorithm. The algorithm must be specified when
178 encryption is enabled because the algorithm is not stored in the disk image.
179 .sp
180 If none of \fB-e\fR, \fB-k\fR, or \fB-T\fR is specified, \fBlofiadm\fR prompts
181 for a passphrase, with a minimum length of eight characters, to be entered .
182 The passphrase is used to derive a symmetric encryption key using PKCS#5 PBKD2.
```



```
183 .RE

185 .sp
186 .ne 2
187 .na
188 \fB\fB-k\fR \fIraw_key_file\fR | \fIwrapped_key_file\fR\fR
189 .ad
190 .sp .6
191 .RS 4n
192 Path to raw or wrapped symmetric encryption key. If a PKCS#11 object is also
193 given with the \fB-T\fR option, then the key is wrapped by that object. If
194 \fB-T\fR is not specified, the key is used raw.
195 .RE

197 .sp
198 .ne 2
199 .na
200 \fB\fB-T\fR \fItoken_key\fR\fR
201 .ad
202 .sp .6
203 .RS 4n
204 The key in a PKCS#11 token to use for the encryption or for unwrapping the key
205 file.
206 .sp
207 If \fB-k\fR is also specified, \fB-T\fR identifies the unwrapping key, which
208 must be an RSA private key.
209 .RE

211 .sp
212 .ne 2
213 .na
214 \fB\fB-e\fR\fR
215 .ad
216 .sp .6
217 .RS 4n
218 Generate an ephemeral symmetric encryption key.
219 .RE

221 .SH OPERANDS
222 .sp
223 .LP
224 The following operands are supported:
225 .sp
226 .ne 2
227 .na
228 \fB\fIcrypto_algorithm\fR\fR
229 .ad
230 .sp .6
231 .RS 4n
232 One of: \fBaes-128-cbc\fR, \fBaes-192-cbc\fR, \fBaes-256-cbc\fR,
233 \fBdes3-cbc\fR, \fBblowfish-cbc\fR.
234 .RE

236 .sp
237 .ne 2
238 .na
239 \fB\fIdevice\fR\fR
240 .ad
241 .sp .6
242 .RS 4n
243 Display the file name associated with the block device \fIdevice\fR.
244 .sp
245 Without arguments, print a list of the current associations. Filenames must be
246 valid absolute pathnames.
247 .sp
248 When a file is added, it is opened for reading or writing by root. Any
```

```
249 restrictions apply (such as restricted root access over \fBNFS\fR). The file is
250 held open until the association is removed. It is not actually accessed until
251 the block device is used, so it will never be written to if the block device is
252 only opened read-only.
253 .RE

255 .sp
256 .ne 2
257 .na
258 \fB\fIfile\fR\fR
259 .ad
260 .sp .6
261 .RS 4n
262 Display the block device associated with \fIfile\fR.
263 .RE

265 .sp
266 .ne 2
267 .na
268 \fB\fIraw_key_file\fR\fR
269 .ad
270 .sp .6
271 .RS 4n
272 Path to a file of the appropriate length, in bits, to use as a raw symmetric
273 encryption key.
274 .RE

276 .sp
277 .ne 2
278 .na
279 \fB\fItoken_key\fR\fR
280 .ad
281 .sp .6
282 .RS 4n
283 PKCS#11 token object in the format:
284 .sp
285 .in +2
286 .nf
287 \fItoken_name\fR:\fImanufacturer_id\fR:\fIserial_number\fR:\fIkey_label\fR
288 .fi
289 .in -2
290 .sp

292 All but the key label are optional and can be empty. For example, to specify a
293 token object with only its key label \fBMylofiKey\fR, use:
294 .sp
295 .in +2
296 .nf
297 -T :::MylofiKey
298 .fi
299 .in -2
300 .sp

302 .RE

304 .sp
305 .ne 2
306 .na
307 \fB\fIwrapped_key_file\fR\fR
308 .ad
309 .sp .6
310 .RS 4n
311 Path to file containing a symmetric encryption key wrapped by the RSA private
312 key specified by \fB-T\fR.
313 .RE
```



```
315 .SH EXAMPLES
316 .LP
317 \fBExample 1 \fRMounting an Existing CD-ROM Image
318 .sp
319 .LP
320 You should ensure that Solaris understands the image before creating the
321 \fBCD\fR. \fBlofi\fR allows you to mount the image and see if it works.

323 .sp
324 .LP
325 This example mounts an existing \fBCD-ROM\fR image (\fBsparc.iso\fR), of the
326 \fBRed Hat 6.0 CD\fR which was downloaded from the Internet. It was created
327 with the \fBmkisofs\fR utility from the Internet.

329 .sp
330 .LP
331 Use \fBlofiadm\fR to attach a block device to it:

333 .sp
334 .in +2
335 .nf
336 # \fBlofiadm -a /home/mike_s/RH6.0/sparc.iso\fR
337 /dev/lofi/1
338 .fi
339 .in -2
340 .sp

342 .sp
343 .LP
344 \fBlofiadm\fR picks the device and prints the device name to the standard
345 output. You can run \fBlofiadm\fR again by issuing the following command:

347 .sp
348 .in +2
349 .nf
350 # \fBlofiadm\fR
351 Block Device     File                           Options
352 /dev/lofi/1      /home/mike_s/RH6.0/sparc.iso   -
353 .fi
354 .in -2
355 .sp

357 .sp
358 .LP
359 Or, you can give it one name and ask for the other, by issuing the following
360 command:

362 .sp
363 .in +2
364 .nf
365 # \fBlofiadm /dev/lofi/1\fR
366 /home/mike_s/RH6.0/sparc.iso
367 .fi
368 .in -2
369 .sp

371 .sp
372 .LP
373 Use the \fBmount\fR command to mount the image:

375 .sp
376 .in +2
377 .nf
378 # \fBmount -F hsfs -o ro /dev/lofi/1 /mnt\fR
379 .fi
380 .in -2
```

```
.sp

.sp
.LP
Check to ensure that Solaris understands the image:

.sp
.in +2
.nf
# \fBdf -k /mnt\fR
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/lofi/1           512418  512418       0   100%    /mnt
# \fBls /mnt\fR
\&./            RedHat/       doc/          ls-lR         rr_moved/
\&../           TRANS.TBL     dosutils/     ls-lR.gz      sbin@
\&.buildlog     bin@          etc@          misc/         tmp/
COPYING       boot/         images/       mnt/          usr@
README        boot.cat*     kernels/      modules/
RPM-PGP-KEY   dev@          lib@          proc/
.fi
.in -2
.sp

.sp
.LP
Solaris can mount the CD-ROM image, and understand the filenames. The image was
created properly, and you can now create the \fBCD-ROM\fR with confidence.

.sp
.LP
As a final step, unmount and detach the images:

.sp
.in +2
.nf
# \fBumount /mnt\fR
# \fBlofiadm -d /dev/lofi/1\fR
# \fBlofiadm\fR
Block Device             File             Options
.fi
.in -2
.sp

.LP
\fBExample 2 \fRMounting a Floppy Image
.sp
.LP
This is similar to the first example.

.sp
.LP
Using \fBlofi\fR to help you mount files that contain floppy images is helpful
if a floppy disk contains a file that you need, but the machine which you are
on does not have a floppy drive. It is also helpful if you do not want to take
the time to use the \fBdd\fR command to copy the image to a floppy.

.sp
.LP
This is an example of getting to \fBMDB\fR floppy for Solaris on an x86
platform:

.sp
.in +2
.nf
# \fBlofiadm -a /export/s28/MDB_s28x_wos/latest/boot.3\fR
/dev/lofi/1
```



```
# \fBmount -F pcfs /dev/lofi/1 /mnt\fR
# \fBls /mnt\fR
\&./            COMMENT.BAT*  RC.D/         SOLARIS.MAP*
\&../           IDENT*        REPLACE.BAT*  X/
APPEND.BAT*   MAKEDIR.BAT*  SOLARIS/
# \fBumount /mnt\fR
# \fBlofiadm -d /export/s28/MDB_s28x_wos/latest/boot.3\fR
.fi
.in -2
.sp

.LP
\fBExample 3 \fRMaking a \fBUFS\fR Filesystem on a File
.sp
.LP
Making a \fBUFS\fR filesystem on a file can be useful, particularly if a test
suite requires a scratch filesystem. It can be painful (or annoying) to have to
repartition a disk just for the test suite, but you do not have to. You can
\fBnewfs\fR a file with \fBlofi\fR

.sp
.LP
Create the file:

.sp
.in +2
.nf
# \fBmkfile 35m /export/home/test\fR
.fi
.in -2
.sp

.sp
.LP
Attach it to a block device. You also get the character device that \fBnewfs\fR
requires, so \fBnewfs\fR that:

.sp
.in +2
.nf
# \fBlofiadm -a /export/home/test\fR
/dev/lofi/1
# \fBnewfs /dev/rlofi/1\fR
newfs: construct a new file system /dev/rlofi/1: (y/n)? \fBy\fR
/dev/rlofi/1:   71638 sectors in 119 cylinders of 1 tracks, 602 sectors
        35.0MB in 8 cyl groups (16 c/g, 4.70MB/g, 2240 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 9664, 19296, 28928, 38560, 48192, 57824, 67456,
.fi
.in -2
.sp

.sp
.LP
Note that \fBufs\fR might not be able to use the entire file. Mount and use the
filesystem:

.sp
.in +2
.nf
# \fBmount /dev/lofi/1 /mnt\fR
# \fBdf -k /mnt\fR
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/lofi/1            33455       9   30101     1%    /mnt
# \fBls /mnt\fR
\&./           ../          lost+found/
```

```
513 # \fBumount /mnt\fR
514 # \fBlofiadm -d /dev/lofi/1\fR
515 .fi
516 .in -2
517 .sp

519 .LP
520 \fBExample 4 \fRCreating a PC (FAT) File System on a Unix File
521 .sp
522 .LP
523 The following series of commands creates a \fBFAT\fR file system on a Unix
524 file. The file is associated with a block device created by \fBlofiadm\fR.

526 .sp
527 .in +2
528 .nf
529 # \fBmkfile 10M /export/test/testfs\fR
530 # \fBlofiadm -a /export/test testfs\fR
531 /dev/lofi/1
532 \fBNote use of\fR rlofi\fB, not\fR lofi\fB, in following command.\fR
533 # \fBmkfs -F pcfs -o nofdisk,size=20480 /dev/rlofi/1\fR
534 \fBConstruct a new FAT file system on /dev/rlofi/1: (y/n)?\fR y
535 # \fBmount -F pcfs /dev/lofi/1 /mnt\fR
536 # \fBcd /mnt\fR
537 # \fBdf -k .\fR
538 Filesystem            kbytes    used   avail capacity  Mounted on
539 /dev/lofi/1            10142       0   10142     0%    /mnt
540 .fi
541 .in -2
542 .sp

544 .LP
545 \fBExample 5 \fRCompressing an Existing CD-ROM Image
546 .sp
547 .LP
548 The following example illustrates compressing an existing CD-ROM image
549 (\fBsolaris.iso\fR), verifying that the image is compressed, and then
550 uncompressing it.

552 .sp
553 .in +2
554 .nf
555 # \fBlofiadm -C gzip /export/home/solaris.iso\fR
556 .fi
557 .in -2
558 .sp

560 .sp
561 .LP
562 Use \fBlofiadm\fR to attach a block device to it:

564 .sp
565 .in +2
566 .nf
567 # \fBlofiadm -a /export/home/solaris.iso\fR
568   /dev/lofi/1
569 .fi
570 .in -2
571 .sp

573 .sp
574 .LP
575 Check if the mapped image is compressed:

577 .sp
578 .in +2
```



```
579 .nf
580 # \fBlofiadm\fR
581 Block Device      File                              Options
582 /dev/lofi/1       /export/home/solaris.iso          Compressed(gzip)
583 /dev/lofi/2       /export/home/regular.iso          -
584 .fi
585 .in -2
586 .sp

588 .sp
589 .LP
590 Unmap the compressed image and uncompress it:

592 .sp
593 .in +2
594 .nf
595 # \fBlofiadm -d /dev/lofi/1\fR
596 # \fBlofiadm -U /export/home/solaris.iso\fR
597 .fi
598 .in -2
599 .sp

601 .LP
602 \fBExample 6 \fRCreating an Encrypted UFS File System on a File
603 .sp
604 .LP
605 This example is similar to the example of making a UFS filesystem on a file,
606 above.

608 .sp
609 .LP
610 Create the file:

612 .sp
613 .in +2
614 .nf
615 # \fBmkfile 35m /export/home/test\fR
616 .fi
617 .in -2
618 .sp

620 .sp
621 .LP
622 Attach the file to a block device and specify that the file image is encrypted.
623 As a result of this command, you obtain the character device, which is
624 subsequently used by \fBnewfs\fR:

626 .sp
627 .in +2
628 .nf
629 # \fBlofiadm -c aes-256-cbc -a /export/home/secrets\fR
630 Enter passphrase: \fBMy-M0th3r;l0v3s_m3+4lw4ys!\fR           (\fBnot echoed\fR)
631 Re-enter passphrase: \fBMy-M0th3r;l0v3s_m3+4lw4ys!\fR       (\fBnot echoed\fR)
632 /dev/lofi/1

634 # \fBnewfs /dev/rlofi/1\fR
635 newfs: construct a new file system /dev/rlofi/1: (y/n)? \fBy\fR
636 /dev/rlofi/1:   71638 sectors in 119 cylinders of 1 tracks, 602 sectors
637        35.0MB in 8 cyl groups (16 c/g, 4.70MB/g, 2240 i/g)
638 super-block backups (for fsck -F ufs -o b=#) at:
639 32, 9664, 19296, 28928, 38560, 48192, 57824, 67456,
640 .fi
641 .in -2
642 .sp

644 .sp
```

```
.LP
The mapped file system shows that encryption is enabled:

.sp
.in +2
.nf
# \fBlofiadm\fR
Block Device    File                    Options
/dev/lofi/1     /export/home/secrets    Encrypted
.fi
.in -2
.sp

.sp
.LP
Mount and use the filesystem:

.sp
.in +2
.nf
# \fBmount /dev/lofi/1 /mnt\fR
# \fBcp moms_secret_*_recipe /mnt\fR
# \fBls /mnt\fR
\&./           moms_secret_cookie_recipe    moms_secret_soup_recipe
\&../          moms_secret_fudge_recipe     moms_secret_stuffing_recipe
lost+found/  moms_secret_meatloaf_recipe  moms_secret_waffle_recipe
# \fBumount /mnt\fR
# \fBlofiadm -d /dev/lofi/1\fR
.fi
.in -2
.sp

.sp
.LP
Subsequent attempts to map the filesystem with the wrong key or the wrong
encryption algorithm will fail:

.sp
.in +2
.nf
# \fBlofiadm -c blowfish-cbc -a /export/home/secrets\fR
Enter passphrase: \fBmommy\fR                              (\fInot echoed\fR)
Re-enter passphrase: \fBmommy\fR                           (\fInot echoed\fR)
lofiadm: could not map file /root/lofi: Invalid argument
# \fBlofiadm\fR
Block Device    File                    Options
#
.fi
.in -2
.sp

.sp
.LP
Attempts to map the filesystem without encryption will succeed, however
attempts to mount and use the filesystem will fail:

.sp
.in +2
.nf
# \fBlofiadm -a /export/home/secrets\fR
/dev/lofi/1
# \fBlofiadm\fR
Block Device    File                    Options
/dev/lofi/1     /export/home/secrets    -
# \fBmount /dev/lofi/1 /mnt\fR
mount: /dev/lofi/1 is not this fstype
```



```
#
.fi
.in -2
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBlofiadm\fR: \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR
and \fBNLSPATH\fR.
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.sp .6
.RS 4n
Successful completion.
.RE

.sp
.ne 2
.na
\fB\fB>0\fR\fR
.ad
.sp .6
.RS 4n
An error occurred.
.RE

.SH SEE ALSO
.sp
.LP
\fBfsck\fR(1M), \fBmount\fR(1M), \fBmount_ufs\fR(1M), \fBnewfs\fR(1M),
\fBattributes\fR(5), \fBlofi\fR(7D), \fBlofs\fR(7FS)
.SH NOTES
.sp
.LP
Just as you would not directly access a disk device that has mounted file
systems, you should not access a file associated with a block device except
through the \fBlofi\fR file driver. It might also be appropriate to ensure that
the file has appropriate permissions to prevent such access.
.sp
.LP
The abilities of \fBlofiadm\fR, and who can use them, are controlled by the
permissions of \fB/dev/lofictl\fR. Read-access allows query operations, such as
listing all the associations. Write-access is required to do any state-changing
operations, like adding an association. As shipped, \fB/dev/lofictl\fR is owned
by \fBroot\fR, in group \fBsys\fR, and mode \fB0644\fR, so all users can do
query operations but only root can change anything. The administrator can give
users write-access, allowing them to add or delete associations, but that is
very likely a security hole and should probably only be given to a trusted
group.
.sp
.LP
When mounting a filesystem image, take care to use appropriate mount options.
In particular, the \fBnosuid\fR mount option might be appropriate for \fBUFS\fR
images whose origin is unknown. Also, some options might not be useful or
appropriate, like \fBlogging\fR or \fBforcedirectio\fR for \fBUFS\fR. For
compatibility purposes, a raw device is also exported along with the block
device. For example, \fBnewfs\fR(1M) requires one.
```

```
777 .sp
778 .LP
779 The output of \fBlofiadm\fR (without arguments) might change in future
780 releases.
```

```
**********************************************************
   74575 Mon Sep 16 15:02:48 2013
new/usr/src/uts/common/io/lofi.c
*** NO COMMENTS ***
**********************************************************
______unchanged_portion_omitted_

 441 /*ARGSUSED*/
 442 static int
 443 lofi_open(dev_t *devp, int flag, int otyp, struct cred *credp)
 444 {
 445         minor_t minor;
 446         struct lofi_state *lsp;

 448         /*
 449          * lofiadm -a /dev/lofi/1 gets us here.
 450          */
 451         if (mutex_owner(&lofi_lock) == curthread)
 452                 return (EINVAL);

 454         mutex_enter(&lofi_lock);

 456         minor = getminor(*devp);

 458         /* master control device */
 459         if (minor == 0) {
 460                 mutex_exit(&lofi_lock);
 461                 return (0);
 462         }

 464         /* otherwise, the mapping should already exist */
 465         lsp = ddi_get_soft_state(lofi_statep, minor);
 466         if (lsp == NULL) {
 467                 mutex_exit(&lofi_lock);
 468                 return (EINVAL);
 469         }

 471         if (lsp->ls_vp == NULL) {
 472                 mutex_exit(&lofi_lock);
 473                 return (ENXIO);
 474         }

 476         if (mark_opened(lsp, otyp) == -1) {
 477                 mutex_exit(&lofi_lock);
 478                 return (EINVAL);
 479         }

 481         if (lsp->ls_readonly && (flag & FWRITE)) {
 482                 mutex_exit(&lofi_lock);
 483                 return (EROFS);
 484         }

 486         mutex_exit(&lofi_lock);
 487         return (0);
 488 }
______unchanged_portion_omitted_

1623 /*
1624  * Find the lofi state for the given filename. We compare by vnode to
1625  * allow the global zone visibility into NGZ lofi nodes.
1626  */
1627 static int
1628 file_to_lofi_nocheck(char *filename, boolean_t readonly,
1629     struct lofi_state **lspp)
1623 file_to_lofi_nocheck(char *filename, struct lofi_state **lspp)
1630 {
```



```
1631         struct lofi_state *lsp;
1632         vnode_t *vp = NULL;
1633         int err = 0;
1634         int rdfiles = 0;

1636         ASSERT(MUTEX_HELD(&lofi_lock));

1638         if ((err = lookupname(filename, UIO_SYSSPACE, FOLLOW,
1639             NULLVPP, &vp)) != 0)
1640                 goto out;

1642         if (vp->v_type == VREG) {
1643                 vnode_t *realvp;
1644                 if (VOP_REALVP(vp, &realvp, NULL) == 0) {
1645                         VN_HOLD(realvp);
1646                         VN_RELE(vp);
1647                         vp = realvp;
1648                 }
1649         }

1651         for (lsp = list_head(&lofi_list); lsp != NULL;
1652             lsp = list_next(&lofi_list, lsp)) {
1653                 if (lsp->ls_vp == vp) {
1654                         if (lspp != NULL)
1655                                 *lspp = lsp;
1656                         if (lsp->ls_readonly) {
1657                                 rdfiles++;
1658                                 /* Skip if '-r' is specified */
1659                                 if (readonly)
1660                                         continue;
1661                         }
1662                         goto out;
1663                 }
1664         }

1666         err = ENOENT;

1668         /*
1669          * If a filename is given as an argument for lofi_unmap, we shouldn't
1670          * allow unmap if there are multiple read-only lofi devices associated
1671          * with this file.
1672          */
1673         if (lspp != NULL) {
1674                 if (rdfiles == 1)
1675                         err = 0;
1676                 else if (rdfiles > 1)
1677                         err = EBUSY;
1678         }

1680 out:
1681         if (vp != NULL)
1682                 VN_RELE(vp);
1683         return (err);
1684 }

1686 /*
1687  * Find the minor for the given filename, checking the zone can access
1688  * it.
1689  */
1690 static int
1691 file_to_lofi(char *filename, boolean_t readonly, struct lofi_state **lspp)
1666 file_to_lofi(char *filename, struct lofi_state **lspp)
1692 {
1693         int err = 0;

1695         ASSERT(MUTEX_HELD(&lofi_lock));
```

```
1697          if ((err = file_to_lofi_nocheck(filename, readonly, lspp)) != 0)
1672          if ((err = file_to_lofi_nocheck(filename, lspp)) != 0)
1698                  return (err);

1700          if ((err = lofi_access(*lspp)) != 0)
1701                  return (err);

1703          return (0);
1704  }
______unchanged_portion_omitted_

2118  /*
2119   * map a file to a minor number. Return the minor number.
2120   */
2121  static int
2122  lofi_map_file(dev_t dev, struct lofi_ioctl *ulip, int pickminor,
2123      int *rvalp, struct cred *credp, int ioctl_flag)
2124  {
2125          minor_t minor = (minor_t)-1;
2126          struct lofi_state *lsp = NULL;
2127          struct lofi_ioctl *klip;
2128          int     error;
2129          struct vnode *vp = NULL;
2130          vattr_t vattr;
2131          int     flag;
2132          dev_t   newdev;
2133          char    namebuf[50];

2135          error = copy_in_lofi_ioctl(ulip, &klip, ioctl_flag);
2136          if (error != 0)
2137                  return (error);

2139          mutex_enter(&lofi_lock);

2141          mutex_enter(&curproc->p_lock);
2142          if ((error = rctl_incr_lofi(curproc, curproc->p_zone, 1)) != 0) {
2143                  mutex_exit(&curproc->p_lock);
2144                  mutex_exit(&lofi_lock);
2145                  free_lofi_ioctl(klip);
2146                  return (error);
2147          }
2148          mutex_exit(&curproc->p_lock);

2150          if (file_to_lofi_nocheck(klip->li_filename, klip->li_readonly,
2151              NULL) == 0) {
2125          if (file_to_lofi_nocheck(klip->li_filename, NULL) == 0) {
2152                  error = EBUSY;
2153                  goto err;
2154          }

2156          if (pickminor) {
2157                  minor = (minor_t)id_allocff_nosleep(lofi_minor_id);
2158                  if (minor == (minor_t)-1) {
2159                          error = EAGAIN;
2160                          goto err;
2161                  }
2162          } else {
2163                  if (ddi_get_soft_state(lofi_statep, klip->li_minor) != NULL) {
2164                          error = EEXIST;
2165                          goto err;
2166                  }

2168                  minor = (minor_t)
2169                      id_alloc_specific_nosleep(lofi_minor_id, klip->li_minor);
2170                  ASSERT(minor != (minor_t)-1);
```



```
2171          }

2173          flag = FREAD | FWRITE | FOFFMAX | FEXCL;
2174          error = vn_open(klip->li_filename, UIO_SYSSPACE, flag, 0, &vp, 0, 0);
2175          if (error) {
2176                  /* try read-only */
2177                  flag &= ~FWRITE;
2178                  error = vn_open(klip->li_filename, UIO_SYSSPACE, flag, 0,
2179                      &vp, 0, 0);
2180                  if (error)
2181                          goto err;
2182          }

2184          if (!V_ISLOFIABLE(vp->v_type)) {
2185                  error = EINVAL;
2186                  goto err;
2187          }

2189          vattr.va_mask = AT_SIZE;
2190          error = VOP_GETATTR(vp, &vattr, 0, credp, NULL);
2191          if (error)
2192                  goto err;

2194          /* the file needs to be a multiple of the block size */
2195          if ((vattr.va_size % DEV_BSIZE) != 0) {
2196                  error = EINVAL;
2197                  goto err;
2198          }

2200          /* lsp alloc+init */

2202          error = ddi_soft_state_zalloc(lofi_statep, minor);
2203          if (error == DDI_FAILURE) {
2204                  error = ENOMEM;
2205                  goto err;
2206          }

2208          lsp = ddi_get_soft_state(lofi_statep, minor);
2209          list_insert_tail(&lofi_list, lsp);

2211          newdev = makedevice(getmajor(dev), minor);
2212          lsp->ls_dev = newdev;
2213          zone_init_ref(&lsp->ls_zone);
2214          zone_hold_ref(curzone, &lsp->ls_zone, ZONE_REF_LOFI);
2215          lsp->ls_uncomp_seg_sz = 0;
2216          lsp->ls_comp_algorithm[0] = '\0';
2217          lsp->ls_crypto_offset = 0;

2219          cv_init(&lsp->ls_vp_cv, NULL, CV_DRIVER, NULL);
2220          mutex_init(&lsp->ls_comp_cache_lock, NULL, MUTEX_DRIVER, NULL);
2221          mutex_init(&lsp->ls_comp_bufs_lock, NULL, MUTEX_DRIVER, NULL);
2222          mutex_init(&lsp->ls_kstat_lock, NULL, MUTEX_DRIVER, NULL);
2223          mutex_init(&lsp->ls_vp_lock, NULL, MUTEX_DRIVER, NULL);

2225          (void) snprintf(namebuf, sizeof (namebuf), "%s_taskq_%d",
2226              LOFI_DRIVER_NAME, minor);
2227          lsp->ls_taskq = taskq_create_proc(namebuf, lofi_taskq_nthreads,
2228              minclsyspri, 1, lofi_taskq_maxalloc, curzone->zone_zsched, 0);

2230          list_create(&lsp->ls_comp_cache, sizeof (struct lofi_comp_cache),
2231              offsetof(struct lofi_comp_cache, lc_list));

2233          /*
2234           * save open mode so file can be closed properly and vnode counts
2235           * updated correctly.
2236           */
```

```
2237         lsp->ls_openflag = flag;

2239         lsp->ls_vp = vp;
2240         lsp->ls_stacked_vp = vp;
2241         /*
2242          * Try to handle stacked lofs vnodes.
2243          */
2244         if (vp->v_type == VREG) {
2245                 vnode_t *realvp;

2247                 if (VOP_REALVP(vp, &realvp, NULL) == 0) {
2248                         /*
2249                          * We need to use the realvp for uniqueness
2250                          * checking, but keep the stacked vp for
2251                          * LOFI_GET_FILENAME display.
2252                          */
2253                         VN_HOLD(realvp);
2254                         lsp->ls_vp = realvp;
2255                 }
2256         }

2258         lsp->ls_vp_size = vattr.va_size;
2259         lsp->ls_vp_comp_size = lsp->ls_vp_size;

2261         lsp->ls_kstat = kstat_create_zone(LOFI_DRIVER_NAME, minor,
2262             NULL, "disk", KSTAT_TYPE_IO, 1, 0, getzoneid());

2264         if (lsp->ls_kstat == NULL) {
2265                 error = ENOMEM;
2266                 goto err;
2267         }

2269         lsp->ls_kstat->ks_lock = &lsp->ls_kstat_lock;
2270         kstat_zone_add(lsp->ls_kstat, GLOBAL_ZONEID);

2272         lsp->ls_readonly = klip->li_readonly;

2274         if ((error = lofi_init_crypto(lsp, klip)) != 0)
2275                 goto err;

2277         if ((error = lofi_init_compress(lsp)) != 0)
2278                 goto err;

2280         fake_disk_geometry(lsp);

2282         /* create minor nodes */

2284         (void) snprintf(namebuf, sizeof (namebuf), "%d", minor);
2285         error = ddi_create_minor_node(lofi_dip, namebuf, S_IFBLK, minor,
2286             DDI_PSEUDO, NULL);
2287         if (error != DDI_SUCCESS) {
2288                 error = ENXIO;
2289                 goto err;
2290         }

2292         (void) snprintf(namebuf, sizeof (namebuf), "%d,raw", minor);
2293         error = ddi_create_minor_node(lofi_dip, namebuf, S_IFCHR, minor,
2294             DDI_PSEUDO, NULL);
2295         if (error != DDI_SUCCESS) {
2296                 /* remove block node */
2297                 (void) snprintf(namebuf, sizeof (namebuf), "%d", minor);
2298                 ddi_remove_minor_node(lofi_dip, namebuf);
2299                 error = ENXIO;
2300                 goto err;
2301         }
```



```
2303         /* create DDI properties */

2305         if ((ddi_prop_update_int64(newdev, lofi_dip, SIZE_PROP_NAME,
2306             lsp->ls_vp_size - lsp->ls_crypto_offset)) != DDI_PROP_SUCCESS) {
2307                 error = EINVAL;
2308                 goto nodeerr;
2309         }

2311         if ((ddi_prop_update_int64(newdev, lofi_dip, NBLOCKS_PROP_NAME,
2312             (lsp->ls_vp_size - lsp->ls_crypto_offset) / DEV_BSIZE))
2313             != DDI_PROP_SUCCESS) {
2314                 error = EINVAL;
2315                 goto nodeerr;
2316         }

2318         if (ddi_prop_update_string(newdev, lofi_dip, ZONE_PROP_NAME,
2319             (char *)curproc->p_zone->zone_name) != DDI_PROP_SUCCESS) {
2320                 error = EINVAL;
2321                 goto nodeerr;
2322         }

2324         kstat_install(lsp->ls_kstat);

2326         mutex_exit(&lofi_lock);

2328         if (rvalp)
2329                 *rvalp = (int)minor;
2330         klip->li_minor = minor;
2331         (void) copy_out_lofi_ioctl(klip, ulip, ioctl_flag);
2332         free_lofi_ioctl(klip);
2333         return (0);

2335 nodeerr:
2336         lofi_free_dev(newdev);
2337 err:
2338         if (lsp != NULL) {
2339                 lofi_destroy(lsp, credp);
2340         } else {
2341                 if (vp != NULL) {
2342                         (void) VOP_CLOSE(vp, flag, 1, 0, credp, NULL);
2343                         VN_RELE(vp);
2344                 }

2346                 if (minor != (minor_t)-1)
2347                         id_free(lofi_minor_id, minor);

2349                 rctl_decr_lofi(curproc->p_zone, 1);
2350         }

2352         mutex_exit(&lofi_lock);
2353         free_lofi_ioctl(klip);
2354         return (error);
2355 }

2357 /*
2358  * unmap a file.
2359  */
2360 static int
2361 lofi_unmap_file(struct lofi_ioctl *ulip, int byfilename,
2362     struct cred *credp, int ioctl_flag)
2363 {
2364         struct lofi_state *lsp;
2365         struct lofi_ioctl *klip;
2366         int err;

2368         err = copy_in_lofi_ioctl(ulip, &klip, ioctl_flag);
```

```
2369         if (err != 0)
2370                 return (err);

2372         mutex_enter(&lofi_lock);
2373         if (byfilename) {
2374                 if ((err = file_to_lofi(klip->li_filename, klip->li_readonly,
2375                     &lsp)) != 0) {
2346                 if ((err = file_to_lofi(klip->li_filename, &lsp)) != 0) {
2376                         mutex_exit(&lofi_lock);
2377                         return (err);
2378                 }
2379         } else if (klip->li_minor == 0) {
2380                 mutex_exit(&lofi_lock);
2381                 free_lofi_ioctl(klip);
2382                 return (ENXIO);
2383         } else {
2384                 lsp = ddi_get_soft_state(lofi_statep, klip->li_minor);
2385         }

2387         if (lsp == NULL || lsp->ls_vp == NULL || lofi_access(lsp) != 0) {
2388                 mutex_exit(&lofi_lock);
2389                 free_lofi_ioctl(klip);
2390                 return (ENXIO);
2391         }

2393         klip->li_minor = getminor(lsp->ls_dev);

2395         /*
2396          * If it's still held open, we'll do one of three things:
2397          *
2398          * If no flag is set, just return EBUSY.
2399          *
2400          * If the 'cleanup' flag is set, unmap and remove the device when
2401          * the last user finishes.
2402          *
2403          * If the 'force' flag is set, then we forcibly close the underlying
2404          * file.  Subsequent operations will fail, and the DKIOCSTATE ioctl
2405          * will return DKIO_DEV_GONE.  When the device is last closed, the
2406          * device will be cleaned up appropriately.
2407          *
2408          * This is complicated by the fact that we may have outstanding
2409          * dispatched I/Os.  Rather than having a single mutex to serialize all
2410          * I/O, we keep a count of the number of outstanding I/O requests
2411          * (ls_vp_iocount), as well as a flag to indicate that no new I/Os
2412          * should be dispatched (ls_vp_closereq).
2413          *
2414          * We set the flag, wait for the number of outstanding I/Os to reach 0,
2415          * and then close the underlying vnode.
2416          */
2417         if (is_opened(lsp)) {
2418                 if (klip->li_force) {
2419                         mutex_enter(&lsp->ls_vp_lock);
2420                         lsp->ls_vp_closereq = B_TRUE;
2421                         /* wake up any threads waiting on dkiocstate */
2422                         cv_broadcast(&lsp->ls_vp_cv);
2423                         while (lsp->ls_vp_iocount > 0)
2424                                 cv_wait(&lsp->ls_vp_cv, &lsp->ls_vp_lock);
2425                         mutex_exit(&lsp->ls_vp_lock);

2427                         goto out;
2428                 } else if (klip->li_cleanup) {
2429                         lsp->ls_cleanup = 1;
2430                         mutex_exit(&lofi_lock);
2431                         free_lofi_ioctl(klip);
2432                         return (0);
2433                 }
```



```
2435                 mutex_exit(&lofi_lock);
2436                 free_lofi_ioctl(klip);
2437                 return (EBUSY);
2438         }

2440 out:
2441         lofi_free_dev(lsp->ls_dev);
2442         lofi_destroy(lsp, credp);

2444         mutex_exit(&lofi_lock);
2445         (void) copy_out_lofi_ioctl(klip, ulip, ioctl_flag);
2446         free_lofi_ioctl(klip);
2447         return (0);
2448 }

2450 /*
2451  * get the filename given the minor number, or the minor number given
2452  * the name.
2453  */
2454 /*ARGSUSED*/
2455 static int
2456 lofi_get_info(dev_t dev, struct lofi_ioctl *ulip, int which,
2457     struct cred *credp, int ioctl_flag)
2458 {
2459         struct lofi_ioctl *klip;
2460         struct lofi_state *lsp;
2461         int     error;

2463         error = copy_in_lofi_ioctl(ulip, &klip, ioctl_flag);
2464         if (error != 0)
2465                 return (error);

2467         switch (which) {
2468         case LOFI_GET_FILENAME:
2469                 if (klip->li_minor == 0) {
2470                         free_lofi_ioctl(klip);
2471                         return (EINVAL);
2472                 }

2474                 mutex_enter(&lofi_lock);
2475                 lsp = ddi_get_soft_state(lofi_statep, klip->li_minor);
2476                 if (lsp == NULL || lofi_access(lsp) != 0) {
2477                         mutex_exit(&lofi_lock);
2478                         free_lofi_ioctl(klip);
2479                         return (ENXIO);
2480                 }

2482                 /*
2483                  * This may fail if, for example, we're trying to look
2484                  * up a zoned NFS path from the global zone.
2485                  */
2486                 if (vnodetopath(NULL, lsp->ls_stacked_vp, klip->li_filename,
2487                     sizeof (klip->li_filename), CRED()) != 0) {
2488                         (void) strlcpy(klip->li_filename, "?",
2489                             sizeof (klip->li_filename));
2490                 }

2492                 klip->li_readonly = lsp->ls_readonly;

2494                 (void) strlcpy(klip->li_algorithm, lsp->ls_comp_algorithm,
2495                     sizeof (klip->li_algorithm));
2496                 klip->li_crypto_enabled = lsp->ls_crypto_enabled;
2497                 mutex_exit(&lofi_lock);
2498                 error = copy_out_lofi_ioctl(klip, ulip, ioctl_flag);
2499                 free_lofi_ioctl(klip);
```

```
2500                 return (error);
2501         case LOFI_GET_MINOR:
2502                 mutex_enter(&lofi_lock);
2503                 error = file_to_lofi(klip->li_filename,
2504                     klip->li_readonly, &lsp);
2472                 error = file_to_lofi(klip->li_filename, &lsp);
2505                 if (error == 0)
2506                         klip->li_minor = getminor(lsp->ls_dev);
2507                 mutex_exit(&lofi_lock);

2509                 if (error == 0)
2510                         error = copy_out_lofi_ioctl(klip, ulip, ioctl_flag);

2512                 free_lofi_ioctl(klip);
2513                 return (error);
2514         case LOFI_CHECK_COMPRESSED:
2515                 mutex_enter(&lofi_lock);
2516                 error = file_to_lofi(klip->li_filename,
2517                     klip->li_readonly, &lsp);
2484                 error = file_to_lofi(klip->li_filename, &lsp);
2518                 if (error != 0) {
2519                         mutex_exit(&lofi_lock);
2520                         free_lofi_ioctl(klip);
2521                         return (error);
2522                 }

2524                 klip->li_minor = getminor(lsp->ls_dev);
2525                 (void) strlcpy(klip->li_algorithm, lsp->ls_comp_algorithm,
2526                     sizeof (klip->li_algorithm));

2528                 mutex_exit(&lofi_lock);
2529                 error = copy_out_lofi_ioctl(klip, ulip, ioctl_flag);
2530                 free_lofi_ioctl(klip);
2531                 return (error);
2532         default:
2533                 free_lofi_ioctl(klip);
2534                 return (EINVAL);
2535         }
2536 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    9322 Mon Sep 16 15:02:50 2013
new/usr/src/uts/common/sys/lofi.h
*** NO COMMENTS ***
**********************************************************
______unchanged_portion_omitted_

 127 struct lofi_ioctl {
 128         uint32_t        li_minor;
 129         boolean_t       li_force;
 130         boolean_t       li_cleanup;
 131         boolean_t       li_readonly;
 132         char    li_filename[MAXPATHLEN];

 134         /* the following fields are required for compression support */
 135         char    li_algorithm[MAXALGLEN];

 137         /* the following fields are required for encryption support */
 138         boolean_t       li_crypto_enabled;
 139         crypto_mech_name_t      li_cipher;      /* for data */
 140         uint32_t        li_key_len;             /* for data */
 141         char            li_key[56];     /* for data: max 448-bit Blowfish key */
 142         crypto_mech_name_t      li_iv_cipher;   /* for iv derivation */
 143         uint32_t        li_iv_len;              /* for iv derivation */
 144         iv_method_t     li_iv_type;             /* for iv derivation */
 145 };
______unchanged_portion_omitted_

 214 struct lofi_state {
 215         vnode_t         *ls_vp;         /* open real vnode */
 216         vnode_t         *ls_stacked_vp; /* open vnode */
 217         kmutex_t        ls_vp_lock;     /* protects ls_vp */
 218         kcondvar_t      ls_vp_cv;       /* signal changes to ls_vp */
 219         uint32_t        ls_vp_iocount;  /* # pending I/O requests */
 220         boolean_t       ls_vp_closereq; /* force close requested */
 221         u_offset_t      ls_vp_size;
 222         uint32_t        ls_blk_open;
 223         uint32_t        ls_chr_open;
 224         uint32_t        ls_lyr_open_count;
 225         int             ls_openflag;
 226         boolean_t       ls_cleanup;     /* cleanup on close */
 227         boolean_t       ls_readonly;
 228         taskq_t         *ls_taskq;
 229         kstat_t         *ls_kstat;
 230         kmutex_t        ls_kstat_lock;
 231         struct dk_geom  ls_dkg;
 232         struct vtoc     ls_vtoc;
 233         struct dk_cinfo ls_ci;
 234         zone_ref_t      ls_zone;
 235         list_node_t     ls_list;        /* all lofis */
 236         dev_t           ls_dev;         /* this node's dev_t */

 238         /* the following fields are required for compression support */
 239         int             ls_comp_algorithm_index; /* idx into compress_table */
 240         char            ls_comp_algorithm[MAXALGLEN];
 241         uint32_t        ls_uncomp_seg_sz; /* sz of uncompressed segment */
 242         uint32_t        ls_comp_index_sz; /* number of index entries */
 243         uint32_t        ls_comp_seg_shift; /* exponent for byte shift */
 244         uint32_t        ls_uncomp_last_seg_sz; /* sz of last uncomp segment */
 245         uint64_t        ls_comp_offbase; /* offset of actual compressed data */
 246         uint64_t        *ls_comp_seg_index; /* array of index entries */
 247         caddr_t         ls_comp_index_data; /* index pages loaded from file */
 248         uint32_t        ls_comp_index_data_sz;
 249         u_offset_t      ls_vp_comp_size; /* actual compressed file size */

 251         /* pre-allocated list of buffers for compressed segment data */
```



```
 252         kmutex_t        ls_comp_bufs_lock;
 253         struct compbuf  *ls_comp_bufs;

 255         /* lock and anchor for compressed segment caching */
 256         kmutex_t        ls_comp_cache_lock;     /* protects ls_comp_cache */
 257         list_t          ls_comp_cache;          /* cached decompressed segs */
 258         uint32_t        ls_comp_cache_count;

 260         /* the following fields are required for encryption support */
 261         boolean_t               ls_crypto_enabled;
 262         u_offset_t              ls_crypto_offset;       /* crypto meta size */
 263         struct crypto_meta      ls_crypto;
 264         crypto_mechanism_t      ls_mech;        /* for data encr/decr */
 265         crypto_key_t            ls_key;         /* for data encr/decr */
 266         crypto_mechanism_t      ls_iv_mech;     /* for iv derivation */
 267         size_t                  ls_iv_len;      /* for iv derivation */
 268         iv_method_t             ls_iv_type;     /* for iv derivation */
 269         kmutex_t                ls_crypto_lock;
 270         crypto_ctx_template_t   ls_ctx_tmpl;

 272 };
______unchanged_portion_omitted_
```