```
**********************************************************
   54666 Sun Jan 17 14:42:13 2016
new/usr/src/cmd/lofiadm/main.c
5857 add -o option to lofiadm
**********************************************************
```

```
   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  *
  26  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
  27  * Copyright (c) 2014 Gary Mills
  28  * Copyright (c) 2016 Andrey Sokolov
  29 #endif /* ! codereview */
  30  */

  32 /*
  33  * lofiadm - administer lofi(7d). Very simple, add and remove file<->device
  34  * associations, and display status. All the ioctls are private between
  35  * lofi and lofiadm, and so are very simple - device information is
  36  * communicated via a minor number.
  37  */

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



```
  62 #include <sys/crypto/ioctladmin.h>
  63 #include "utils.h"
  64 #include <LzmaEnc.h>

  66 /* Only need the IV len #defines out of these files, nothing else. */
  67 #include <aes/aes_impl.h>
  68 #include <des/des_impl.h>
  69 #include <blowfish/blowfish_impl.h>

  71 static const char USAGE[] =
  72         "Usage: %s [-r] -a file [ device ]\n"
  73         "       %s [-r] [-o] -c crypto_algorithm -a file [device]\n"
  28         "       %s [-r] -c crypto_algorithm -a file [device]\n"
  74         "       %s [-r] -c crypto_algorithm -k raw_key_file -a file [device]\n"
  75         "       %s [-r] -c crypto_algorithm -T [token]:[manuf]:[serial]:key "
  76         "-a file [device]\n"
  77         "       %s [-r] -c crypto_algorithm -T [token]:[manuf]:[serial]:key "
  78         "-k wrapped_key_file -a file [device]\n"
  79         "       %s [-r] -c crypto_algorithm -e -a file [device]\n"
  80         "       %s -d file | device\n"
  81         "       %s -C [gzip|gzip-6|gzip-9|lzma] [-s segment_size] file\n"
  82         "       %s -U file\n"
  83         "       %s [ file | device ]\n";

  85 typedef struct token_spec {
  86         char    *name;
  87         char    *mfr;
  88         char    *serno;
  89         char    *key;
  90 } token_spec_t;
_____unchanged_portion_omitted_

 836 /*
 837  * PBE the passphrase into a raw key
 838  */
 839 static void
 840 getkeyfromuser(mech_alias_t *cipher, char **raw_key, size_t *raw_key_sz,
 841     boolean_t with_confirmation)
 795 getkeyfromuser(mech_alias_t *cipher, char **raw_key, size_t *raw_key_sz)
 842 {
 843         CK_SESSION_HANDLE sess;
 844         CK_RV   rv;
 845         char    *pass = NULL;
 846         size_t  passlen = 0;
 847         void    *salt = NULL;   /* don't use NULL, see note on salt below */
 848         size_t  saltlen = 0;
 849         CK_KEY_TYPE ktype;
 850         void    *kvalue;
 851         size_t  klen;

 853         /* did init_crypto find a slot that supports this cipher? */
 854         if (cipher->slot == (CK_SLOT_ID)-1 || cipher->max_keysize == 0) {
 855                 rv = CKR_MECHANISM_INVALID;
 856                 goto cleanup;
 857         }

 859         rv = pkcs11_mech2keytype(cipher->type, &ktype);
 860         if (rv != CKR_OK)
 861                 goto cleanup;

 863         /*
 864          * use the passphrase to generate a PBE PKCS#5 secret key and
 865          * retrieve the raw key data to eventually pass it to the kernel;
 866          */
 867         rv = C_OpenSession(cipher->slot, CKF_SERIAL_SESSION, NULL, NULL, &sess);
 868         if (rv != CKR_OK)
```

```
 869                 goto cleanup;

 871         /* get user passphrase with 8 byte minimum */
 872         if (pkcs11_get_pass(NULL, &pass, &passlen, MIN_PASSLEN,
 873             with_confirmation) < 0) {
 826         if (pkcs11_get_pass(NULL, &pass, &passlen, MIN_PASSLEN, B_TRUE) < 0) {
 874                 die(gettext("passphrases do not match\n"));
 875         }

 877         /*
 878          * salt should not be NULL, or else pkcs11_PasswdToKey() will
 879          * complain about CKR_MECHANISM_PARAM_INVALID; the following is
 880          * to make up for not having a salt until a proper one is used
 881          */
 882         salt = pass;
 883         saltlen = passlen;

 885         klen = cipher->max_keysize;
 886         rv = pkcs11_PasswdToKey(sess, pass, passlen, salt, saltlen, ktype,
 887             cipher->max_keysize, &kvalue, &klen);

 889         (void) C_CloseSession(sess);

 891         if (rv != CKR_OK) {
 892                 goto cleanup;
 893         }

 895         /* assert(klen == cipher->max_keysize); */
 896         *raw_key_sz = klen;
 897         *raw_key = (char *)kvalue;
 898         return;

 900 cleanup:
 901         die(gettext("failed to generate %s key from passphrase: %s"),
 902             cipher->alias, pkcs11_strerror(rv));
 903 }
_____unchanged_portion_omitted_

1802 int
1803 main(int argc, char *argv[])
1804 {
1805         int     lfd;
1806         int     c;
1807         const char *devicename = NULL;
1808         const char *filename = NULL;
1809         const char *algname = COMPRESS_ALGORITHM;
1810         int     openflag;
1811         int     minor;
1812         int     compress_index;
1813         uint32_t segsize = SEGSIZE;
1814         static char *lofictl = "/dev/" LOFI_CTL_NAME;
1815         boolean_t force = B_FALSE;
1816         const char *pname;
1817         boolean_t errflag = B_FALSE;
1818         boolean_t addflag = B_FALSE;
1819         boolean_t rdflag = B_FALSE;
1820         boolean_t deleteflag = B_FALSE;
1821         boolean_t ephflag = B_FALSE;
1822         boolean_t compressflag = B_FALSE;
1823         boolean_t uncompressflag = B_FALSE;
1824         /* the next two work together for -c, -k, -T, -e options only */
1825         boolean_t need_crypto = B_FALSE;        /* if any -c, -k, -T, -e */
1826         boolean_t cipher_only = B_TRUE;         /* if -c only */
1827         boolean_t with_confirmation = B_TRUE;
1828 #endif /* ! codereview */
1829         const char *keyfile = NULL;
```



```
1830         mech_alias_t *cipher = NULL;
1831         token_spec_t *token = NULL;
1832         char    *rkey = NULL;
1833         size_t  rksz = 0;
1834         char realfilename[MAXPATHLEN];

1836         pname = getpname(argv[0]);

1838         (void) setlocale(LC_ALL, "");
1839         (void) textdomain(TEXT_DOMAIN);

1841         while ((c = getopt(argc, argv, "a:c:Cd:efk:ors:T:U")) != EOF) {
1780         while ((c = getopt(argc, argv, "a:c:Cd:efk:o:rs:T:U")) != EOF) {
1842                 switch (c) {
1843                 case 'a':
1844                         addflag = B_TRUE;
1845                         if ((filename = realpath(optarg, realfilename)) == NULL)
1846                                 die("%s", optarg);
1847                         if (((argc - optind) > 0) && (*argv[optind] != '-')) {
1848                                 /* optional device */
1849                                 devicename = argv[optind];
1850                                 optind++;
1851                         }
1852                         break;
1853                 case 'C':
1854                         compressflag = B_TRUE;
1855                         if (((argc - optind) > 1) && (*argv[optind] != '-')) {
1856                                 /* optional algorithm */
1857                                 algname = argv[optind];
1858                                 optind++;
1859                         }
1860                         check_algorithm_validity(algname, &compress_index);
1861                         break;
1862                 case 'c':
1863                         /* is the chosen cipher allowed? */
1864                         if ((cipher = ciph2mech(optarg)) == NULL) {
1865                                 errflag = B_TRUE;
1866                                 warn(gettext("cipher %s not allowed\n"),
1867                                     optarg);
1868                         }
1869                         need_crypto = B_TRUE;
1870                         /* cipher_only is already set */
1871                         break;
1872                 case 'd':
1873                         deleteflag = B_TRUE;
1874                         minor = name_to_minor(optarg);
1875                         if (minor != 0)
1876                                 devicename = optarg;
1877                         else {
1878                                 if ((filename = realpath(optarg,
1879                                     realfilename)) == NULL)
1880                                         die("%s", optarg);
1881                         }
1882                         break;
1883                 case 'e':
1884                         ephflag = B_TRUE;
1885                         need_crypto = B_TRUE;
1886                         cipher_only = B_FALSE;  /* need to unset cipher_only */
1887                         break;
1888                 case 'f':
1889                         force = B_TRUE;
1890                         break;
1891                 case 'k':
1892                         keyfile = optarg;
1893                         need_crypto = B_TRUE;
1894                         cipher_only = B_FALSE;  /* need to unset cipher_only */
```

```
1895                        break;
1896                case 'r':
1897                        rdflag = B_TRUE;
1898                        break;
1899                case 's':
1900                        segsize = convert_to_num(optarg);
1901                        if (segsize < DEV_BSIZE || !ISP2(segsize))
1902                                die(gettext("segment size %s is invalid "
1903                                    "or not a multiple of minimum block "
1904                                    "size %ld\n"), optarg, DEV_BSIZE);
1905                        break;
1906                case 'T':
1907                        if ((token = parsetoken(optarg)) == NULL) {
1908                                errflag = B_TRUE;
1909                                warn(
1910                                    gettext("invalid token key specifier %s\n"),
1911                                    optarg);
1912                        }
1913                        need_crypto = B_TRUE;
1914                        cipher_only = B_FALSE;  /* need to unset cipher_only */
1915                        break;
1916                case 'U':
1917                        uncompressflag = B_TRUE;
1918                        break;
1919                case 'o':
1920                        with_confirmation = B_FALSE;
1921                        break;
1922 #endif /* ! codereview */
1923                case '?':
1924                default:
1925                        errflag = B_TRUE;
1926                        break;
1927                }
1928        }

1930        /* Check for mutually exclusive combinations of options */
1931        if (errflag ||
1932            (addflag && deleteflag) ||
1933            (rdflag && !addflag) ||
1934            (!addflag && need_crypto) ||
1935            (!with_confirmation && (!cipher_only || !need_crypto)) ||
1936 #endif /* ! codereview */
1937            ((compressflag || uncompressflag) && (addflag || deleteflag)))
1938                usage(pname);

1940        /* ephemeral key, and key from either file or token are incompatible */
1941        if (ephflag && (keyfile != NULL || token != NULL)) {
1942                die(gettext("ephemeral key cannot be used with keyfile"
1943                    " or token key\n"));
1944        }

1946        /*
1947         * "-c" but no "-k", "-T", "-e", or "-T -k" means derive key from
1948         * command line passphrase
1949         */

1951        switch (argc - optind) {
1952        case 0: /* no more args */
1953                if (compressflag || uncompressflag)     /* needs filename */
1954                        usage(pname);
1955                break;
1956        case 1:
1957                if (addflag || deleteflag)
1958                        usage(pname);
1959                /* one arg means compress/uncompress the file ... */
1960                if (compressflag || uncompressflag) {
```



```
1961                        if ((filename = realpath(argv[optind],
1962                            realfilename)) == NULL)
1963                                die("%s", argv[optind]);
1964                /* ... or without options means print the association */
1965                } else {
1966                        minor = name_to_minor(argv[optind]);
1967                        if (minor != 0)
1968                                devicename = argv[optind];
1969                        else {
1970                                if ((filename = realpath(argv[optind],
1971                                    realfilename)) == NULL)
1972                                        die("%s", argv[optind]);
1973                        }
1974                }
1975                break;
1976        default:
1977                usage(pname);
1978                break;
1979        }

1981        if (addflag || compressflag || uncompressflag)
1982                check_file_validity(filename);

1984        if (filename && !valid_abspath(filename))
1985                exit(E_ERROR);

1987        /*
1988         * Here, we know the arguments are correct, the filename is an
1989         * absolute path, it exists and is a regular file. We don't yet
1990         * know that the device name is ok or not.
1991         */

1993        openflag = O_EXCL;
1994        if (addflag || deleteflag || compressflag || uncompressflag)
1995                openflag |= O_RDWR;
1996        else
1997                openflag |= O_RDONLY;
1998        lfd = open(lofictl, openflag);
1999        if (lfd == -1) {
2000                if ((errno == EPERM) || (errno == EACCES)) {
2001                        die(gettext("you do not have permission to perform "
2002                            "that operation.\n"));
2003                } else {
2004                        die(gettext("open: %s"), lofictl);
2005                }
2006                /*NOTREACHED*/
2007        }

2009        /*
2010         * No passphrase is needed for ephemeral key, or when key is
2011         * in a file and not wrapped by another key from a token.
2012         * However, a passphrase is needed in these cases:
2013         * 1. cipher with no ephemeral key, key file, or token,
2014         *    in which case the passphrase is used to build the key
2015         * 2. token with an optional cipher or optional key file,
2016         *    in which case the passphrase unlocks the token
2017         * If only the cipher is specified, reconfirm the passphrase
2018         * to ensure the user hasn't mis-entered it.  Otherwise, the
2019         * token will enforce the token passphrase.
2020         */
2021        if (need_crypto) {
2022                CK_SESSION_HANDLE       sess;

2024                /* pick a cipher if none specified */
2025                if (cipher == NULL)
2026                        cipher = DEFAULT_CIPHER;
```

```
2028                 if (!kernel_cipher_check(cipher))
2029                         die(gettext(
2030                             "use \"cryptoadm list -m\" to find available "
2031                             "mechanisms\n"));

2033                 init_crypto(token, cipher, &sess);

2035                 if (cipher_only) {
2036                         getkeyfromuser(cipher, &rkey, &rksz, with_confirmation);
1858                         getkeyfromuser(cipher, &rkey, &rksz);
2037                 } else if (token != NULL) {
2038                         getkeyfromtoken(sess, token, keyfile, cipher,
2039                             &rkey, &rksz);
2040                 } else {
2041                         /* this also handles ephemeral keys */
2042                         getkeyfromfile(keyfile, cipher, &rkey, &rksz);
2043                 }

2045                 end_crypto(sess);
2046         }

2048         /*
2049          * Now to the real work.
2050          */
2051         if (addflag)
2052                 add_mapping(lfd, devicename, filename, cipher, rkey, rksz,
2053                     rdflag);
2054         else if (compressflag)
2055                 lofi_compress(&lfd, filename, compress_index, segsize);
2056         else if (uncompressflag)
2057                 lofi_uncompress(lfd, filename);
2058         else if (deleteflag)
2059                 delete_mapping(lfd, devicename, filename, force);
2060         else if (filename || devicename)
2061                 print_one_mapping(lfd, devicename, filename);
2062         else
2063                 print_mappings(lfd);

2065         if (lfd != -1)
2066                 (void) close(lfd);
2067         closelib();
2068         return (E_SUCCESS);
2069 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   18522 Sun Jan 17 14:42:14 2016
new/usr/src/man/man1m/lofiadm.1m
5857 add -o option to lofiadm
**********************************************************
'\" te
.\" Copyright (c) 2016 Andrey Sokolov
#endif /* ! codereview */
.\" Copyright 2013 Nexenta Systems, Inc. All rights reserved.
.\" 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"
.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]
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] [\fB-o\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-a\fR \fI
 \fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-a\fR \fIfile\fR [\f
.fi

.LP
.nf
\fBlofiadm\fR [\fB-r\fR] \fB-c\fR \fIcrypto_algorithm\fR \fB-k\fR \fIraw_key_fil
.fi

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

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

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

.LP
.nf
\fBlofiadm\fR \fB-d\fR \fIfile\fR | \fIdevice\fR
.fi

.LP
.nf
\fBlofiadm\fR \fB-U\fR \fIfile\fR
.fi

.LP
.nf
```



```
\fBlofiadm\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.

In the global zone, \fBlofiadm\fR can be used on both the global
zone devices and all devices owned by other non-global zones on the system.
.sp
.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).
.sp
\fIlzma\fR stands for the LZMA (Lempel-Ziv-Markov) compression algorithm.
.sp
Note that you cannot write to a compressed file, nor can you mount a compressed
file read/write.
.RE
```

```
.sp
.ne 2
.na
\fB\fB-d\fR \fIfile\fR | \fIdevice\fR\fR
.ad
.sp .6
.RS 4n
Remove an association by \fIfile\fR or \fIdevice\fR name, if the associated
block device is not busy, and deallocates the block device.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR
.ad
.sp .6
.RS 4n
If the \fB-o\fR option is specified lofiadm will prompt for a passphrase once.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-r\fR
.ad
.sp .6
.RS 4n
If the \fB-r\fR option is specified before the \fB-a\fR option, the
\fIdevice\fR will be opened read-only.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIsegment_size\fR\fR
.ad
.sp .6
.RS 4n
The segment size to use to divide the file being compressed. \fIsegment_size\fR
can be an integer multiple of 512.
.RE

.sp
.ne 2
.na
\fB\fB-U\fR \fIfile\fR\fR
.ad
.sp .6
.RS 4n
Uncompress a compressed file.
.RE

.sp
.LP
The following options are used when the file is encrypted:
.sp
.ne 2
.na
\fB\fB-c\fR \fIcrypto_algorithm\fR\fR
.ad
.sp .6
.RS 4n
Select the encryption algorithm. The algorithm must be specified when
```



```
encryption is enabled because the algorithm is not stored in the disk image.
.sp
If none of \fB-e\fR, \fB-k\fR, or \fB-T\fR is specified, \fBlofiadm\fR prompts
for a passphrase, with a minimum length of eight characters, to be entered .
The passphrase is used to derive a symmetric encryption key using PKCS#5 PBKD2.
.RE

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

.sp
.ne 2
.na
\fB\fB-T\fR \fItoken_key\fR\fR
.ad
.sp .6
.RS 4n
The key in a PKCS#11 token to use for the encryption or for unwrapping the key
file.
.sp
If \fB-k\fR is also specified, \fB-T\fR identifies the unwrapping key, which
must be an RSA private key.
.RE

.sp
.ne 2
.na
\fB\fB-e\fR\fR
.ad
.sp .6
.RS 4n
Generate an ephemeral symmetric encryption key.
.RE

.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIcrypto_algorithm\fR\fR
.ad
.sp .6
.RS 4n
One of: \fBaes-128-cbc\fR, \fBaes-192-cbc\fR, \fBaes-256-cbc\fR,
\fBdes3-cbc\fR, \fBblowfish-cbc\fR.
.RE

.sp
.ne 2
.na
\fB\fIdevice\fR\fR
.ad
.sp .6
.RS 4n
Display the file name associated with the block device \fIdevice\fR.
```

```
256 .sp
257 Without arguments, print a list of the current associations. Filenames must be
258 valid absolute pathnames.
259 .sp
260 When a file is added, it is opened for reading or writing by root. Any
261 restrictions apply (such as restricted root access over \fBNFS\fR). The file is
262 held open until the association is removed. It is not actually accessed until
263 the block device is used, so it will never be written to if the block device is
264 only opened read-only.

266 Note that the filename may appear as "?" if it is not possible to resolve the
267 path in the current context (for example, if it’s an NFS path in a non-global
268 zone).
269 .RE

271 .sp
272 .ne 2
273 .na
274 \fB\fIfile\fR\fR
275 .ad
276 .sp .6
277 .RS 4n
278 Display the block device associated with \fIfile\fR.
279 .RE

281 .sp
282 .ne 2
283 .na
284 \fB\fIraw_key_file\fR\fR
285 .ad
286 .sp .6
287 .RS 4n
288 Path to a file of the appropriate length, in bits, to use as a raw symmetric
289 encryption key.
290 .RE

292 .sp
293 .ne 2
294 .na
295 \fB\fItoken_key\fR\fR
296 .ad
297 .sp .6
298 .RS 4n
299 PKCS#11 token object in the format:
300 .sp
301 .in +2
302 .nf
303 \fItoken_name\fR:\fImanufacturer_id\fR:\fIserial_number\fR:\fIkey_label\fR
304 .fi
305 .in -2
306 .sp

308 All but the key label are optional and can be empty. For example, to specify a
309 token object with only its key label \fBMylofiKey\fR, use:
310 .sp
311 .in +2
312 .nf
313 -T :::MylofiKey
314 .fi
315 .in -2
316 .sp

318 .RE

320 .sp
321 .ne 2
```



```
322 .na
323 \fB\fIwrapped_key_file\fR\fR
324 .ad
325 .sp .6
326 .RS 4n
327 Path to file containing a symmetric encryption key wrapped by the RSA private
328 key specified by \fB-T\fR.
329 .RE

331 .SH EXAMPLES
332 .LP
333 \fBExample 1 \fRMounting an Existing CD-ROM Image
334 .sp
335 .LP
336 You should ensure that Solaris understands the image before creating the
337 \fBCD\fR. \fBlofi\fR allows you to mount the image and see if it works.

339 .sp
340 .LP
341 This example mounts an existing \fBCD-ROM\fR image (\fBsparc.iso\fR), of the
342 \fBRed Hat 6.0 CD\fR which was downloaded from the Internet. It was created
343 with the \fBmkisofs\fR utility from the Internet.

345 .sp
346 .LP
347 Use \fBlofiadm\fR to attach a block device to it:

349 .sp
350 .in +2
351 .nf
352 # \fBlofiadm -a /home/mike_s/RH6.0/sparc.iso\fR
353 /dev/lofi/1
354 .fi
355 .in -2
356 .sp

358 .sp
359 .LP
360 \fBlofiadm\fR picks the device and prints the device name to the standard
361 output. You can run \fBlofiadm\fR again by issuing the following command:

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

373 .sp
374 .LP
375 Or, you can give it one name and ask for the other, by issuing the following
376 command:

378 .sp
379 .in +2
380 .nf
381 # \fBlofiadm /dev/lofi/1\fR
382 /home/mike_s/RH6.0/sparc.iso
383 .fi
384 .in -2
385 .sp

387 .sp
```

```
388 .LP
389 Use the \fBmount\fR command to mount the image:

391 .sp
392 .in +2
393 .nf
394 # \fBmount -F hsfs -o ro /dev/lofi/1 /mnt\fR
395 .fi
396 .in -2
397 .sp

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

403 .sp
404 .in +2
405 .nf
406 # \fBdf -k /mnt\fR
407 Filesystem            kbytes    used   avail capacity  Mounted on
408 /dev/lofi/1           512418  512418       0   100%    /mnt
409 # \fBls /mnt\fR
410 \&./           RedHat/       doc/          ls-lR         rr_moved/
411 \&../          TRANS.TBL     dosutils/     ls-lR.gz      sbin@
412 \&.buildlog    bin@          etc@          misc/         tmp/
413 COPYING       boot/         images/       mnt/          usr@
414 README        boot.cat*     kernels/      modules/
415 RPM-PGP-KEY   dev@          lib@          proc/
416 .fi
417 .in -2
418 .sp

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

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

429 .sp
430 .in +2
431 .nf
432 # \fBumount /mnt\fR
433 # \fBlofiadm -d /dev/lofi/1\fR
434 # \fBlofiadm\fR
435 Block Device             File             Options
436 .fi
437 .in -2
438 .sp

440 .LP
441 \fBExample 2 \fRMounting a Floppy Image
442 .sp
443 .LP
444 This is similar to the first example.

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

453 .sp
```



```
454 .LP
455 This is an example of getting to \fBMDB\fR floppy for Solaris on an x86
456 platform:

458 .sp
459 .in +2
460 .nf
461 # \fBlofiadm -a /export/s28/MDB_s28x_wos/latest/boot.3\fR
462 /dev/lofi/1
463 # \fBmount -F pcfs /dev/lofi/1 /mnt\fR
464 # \fBls /mnt\fR
465 \&./           COMMENT.BAT*  RC.D/         SOLARIS.MAP*
466 \&../          IDENT*        REPLACE.BAT*  X/
467 APPEND.BAT*   MAKEDIR.BAT*  SOLARIS/
468 # \fBumount /mnt\fR
469 # \fBlofiadm -d /export/s28/MDB_s28x_wos/latest/boot.3\fR
470 .fi
471 .in -2
472 .sp

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

483 .sp
484 .LP
485 Create the file:

487 .sp
488 .in +2
489 .nf
490 # \fBmkfile 35m /export/home/test\fR
491 .fi
492 .in -2
493 .sp

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

500 .sp
501 .in +2
502 .nf
503 # \fBlofiadm -a /export/home/test\fR
504 /dev/lofi/1
505 # \fBnewfs /dev/rlofi/1\fR
506 newfs: construct a new file system /dev/rlofi/1: (y/n)? \fBy\fR
507 /dev/rlofi/1:   71638 sectors in 119 cylinders of 1 tracks, 602 sectors
508         35.0MB in 8 cyl groups (16 c/g, 4.70MB/g, 2240 i/g)
509 super-block backups (for fsck -F ufs -o b=#) at:
510  32, 9664, 19296, 28928, 38560, 48192, 57824, 67456,
511 .fi
512 .in -2
513 .sp

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

```
520 .sp
521 .in +2
522 .nf
523 # \fBmount /dev/lofi/1 /mnt\fR
524 # \fBdf -k /mnt\fR
525 Filesystem            kbytes    used   avail capacity  Mounted on
526 /dev/lofi/1            33455       9   30101     1%    /mnt
527 # \fBls /mnt\fR
528 \&./          ../          lost+found/
529 # \fBumount /mnt\fR
530 # \fBlofiadm -d /dev/lofi/1\fR
531 .fi
532 .in -2
533 .sp

535 .LP
536 \fBExample 4 \fRCreating a PC (FAT) File System on a Unix File
537 .sp
538 .LP
539 The following series of commands creates a \fBFAT\fR file system on a Unix
540 file. The file is associated with a block device created by \fBlofiadm\fR.

542 .sp
543 .in +2
544 .nf
545 # \fBmkfile 10M /export/test/testfs\fR
546 # \fBlofiadm -a /export/test testfs\fR
547 /dev/lofi/1
548 \fBNote use of\fR rlofi\fB, not\fR lofi\fB, in following command.\fR
549 # \fBmkfs -F pcfs -o nofdisk,size=20480 /dev/rlofi/1\fR
550 \fBConstruct a new FAT file system on /dev/rlofi/1: (y/n)?\fR y
551 # \fBmount -F pcfs /dev/lofi/1 /mnt\fR
552 # \fBcd /mnt\fR
553 # \fBdf -k .\fR
554 Filesystem            kbytes    used   avail capacity  Mounted on
555 /dev/lofi/1            10142       0   10142     0%    /mnt
556 .fi
557 .in -2
558 .sp

560 .LP
561 \fBExample 5 \fRCompressing an Existing CD-ROM Image
562 .sp
563 .LP
564 The following example illustrates compressing an existing CD-ROM image
565 (\fBsolaris.iso\fR), verifying that the image is compressed, and then
566 uncompressing it.

568 .sp
569 .in +2
570 .nf
571 # \fBlofiadm -C gzip /export/home/solaris.iso\fR
572 .fi
573 .in -2
574 .sp

576 .sp
577 .LP
578 Use \fBlofiadm\fR to attach a block device to it:

580 .sp
581 .in +2
582 .nf
583 # \fBlofiadm -a /export/home/solaris.iso\fR
584   /dev/lofi/1
585 .fi
```



```
586 .in -2
587 .sp

589 .sp
590 .LP
591 Check if the mapped image is compressed:

593 .sp
594 .in +2
595 .nf
596 # \fBlofiadm\fR
597 Block Device      File                              Options
598 /dev/lofi/1       /export/home/solaris.iso        Compressed(gzip)
599 /dev/lofi/2       /export/home/regular.iso        -
600 .fi
601 .in -2
602 .sp

604 .sp
605 .LP
606 Unmap the compressed image and uncompress it:

608 .sp
609 .in +2
610 .nf
611 # \fBlofiadm -d /dev/lofi/1\fR
612 # \fBlofiadm -U /export/home/solaris.iso\fR
613 .fi
614 .in -2
615 .sp

617 .LP
618 \fBExample 6 \fRCreating an Encrypted UFS File System on a File
619 .sp
620 .LP
621 This example is similar to the example of making a UFS filesystem on a file,
622 above.

624 .sp
625 .LP
626 Create the file:

628 .sp
629 .in +2
630 .nf
631 # \fBmkfile 35m /export/home/test\fR
632 .fi
633 .in -2
634 .sp

636 .sp
637 .LP
638 Attach the file to a block device and specify that the file image is encrypted.
639 As a result of this command, you obtain the character device, which is
640 subsequently used by \fBnewfs\fR:

642 .sp
643 .in +2
644 .nf
645 # \fBlofiadm -c aes-256-cbc -a /export/home/secrets\fR
646 Enter passphrase: \fBMy-M0th3r;l0v3s_m3+4lw4ys!\fR           (\fBnot echoed\fR)
647 Re-enter passphrase: \fBMy-M0th3r;l0v3s_m3+4lw4ys!\fR       (\fBnot echoed\fR)
648 /dev/lofi/1

650 # \fBnewfs /dev/rlofi/1\fR
651 newfs: construct a new file system /dev/rlofi/1: (y/n)? \fBy\fR
```

```
652 /dev/rlofi/1:   71638 sectors in 119 cylinders of 1 tracks, 602 sectors
653        35.0MB in 8 cyl groups (16 c/g, 4.70MB/g, 2240 i/g)
654 super-block backups (for fsck -F ufs -o b=#) at:
655 32, 9664, 19296, 28928, 38560, 48192, 57824, 67456,
656 .fi
657 .in -2
658 .sp

660 .sp
661 .LP
662 The mapped file system shows that encryption is enabled:

664 .sp
665 .in +2
666 .nf
667 # \fBlofiadm\fR
668 Block Device    File                    Options
669 /dev/lofi/1     /export/home/secrets    Encrypted
670 .fi
671 .in -2
672 .sp

674 .sp
675 .LP
676 Mount and use the filesystem:

678 .sp
679 .in +2
680 .nf
681 # \fBmount /dev/lofi/1 /mnt\fR
682 # \fBcp moms_secret_*_recipe /mnt\fR
683 # \fBls /mnt\fR
684 \&./           moms_secret_cookie_recipe    moms_secret_soup_recipe
685 \&../          moms_secret_fudge_recipe     moms_secret_stuffing_recipe
686 lost+found/  moms_secret_meatloaf_recipe  moms_secret_waffle_recipe
687 # \fBumount /mnt\fR
688 # \fBlofiadm -d /dev/lofi/1\fR
689 .fi
690 .in -2
691 .sp

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

698 .sp
699 .in +2
700 .nf
701 # \fBlofiadm -c blowfish-cbc -a /export/home/secrets\fR
702 Enter passphrase: \fBmommy\fR                              (\fInot echoed\fR)
703 Re-enter passphrase: \fBmommy\fR                           (\fInot echoed\fR)
704 lofiadm: could not map file /root/lofi: Invalid argument
705 # \fBlofiadm\fR
706 Block Device    File                    Options
707 #
708 .fi
709 .in -2
710 .sp

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

717 .sp
```



```
718 .in +2
719 .nf
720 # \fBlofiadm -a /export/home/secrets\fR
721 /dev/lofi/1
722 # \fBlofiadm\fR
723 Block Device    File                    Options
724 /dev/lofi/1     /export/home/secrets    -
725 # \fBmount /dev/lofi/1 /mnt\fR
726 mount: /dev/lofi/1 is not this fstype
727 #
728 .fi
729 .in -2
730 .sp

732 .SH ENVIRONMENT VARIABLES
623 .sp
733 .LP
734 See \fBenviron\fR(5) for descriptions of the following environment variables
735 that affect the execution of \fBlofiadm\fR: \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR
736 and \fBNLSPATH\fR.
737 .SH EXIT STATUS
629 .sp
738 .LP
739 The following exit values are returned:
740 .sp
741 .ne 2
742 .na
743 \fB\fB0\fR\fR
744 .ad
745 .sp .6
746 .RS 4n
747 Successful completion.
748 .RE

750 .sp
751 .ne 2
752 .na
753 \fB\fB>0\fR\fR
754 .ad
755 .sp .6
756 .RS 4n
757 An error occurred.
758 .RE

760 .SH SEE ALSO
653 .sp
761 .LP
762 \fBfsck\fR(1M), \fBmount\fR(1M), \fBmount_ufs\fR(1M), \fBnewfs\fR(1M),
763 \fBattributes\fR(5), \fBlofi\fR(7D), \fBlofs\fR(7FS)
764 .SH NOTES
658 .sp
765 .LP
766 Just as you would not directly access a disk device that has mounted file
767 systems, you should not access a file associated with a block device except
768 through the \fBlofi\fR file driver. It might also be appropriate to ensure that
769 the file has appropriate permissions to prevent such access.
770 .sp
771 .LP
772 The abilities of \fBlofiadm\fR, and who can use them, are controlled by the
773 permissions of \fB/dev/lofictl\fR. Read-access allows query operations, such as
774 listing all the associations. Write-access is required to do any state-changing
775 operations, like adding an association. As shipped, \fB/dev/lofictl\fR is owned
776 by \fBroot\fR, in group \fBsys\fR, and mode \fB0644\fR, so all users can do
777 query operations but only root can change anything. The administrator can give
778 users write-access, allowing them to add or delete associations, but that is
779 very likely a security hole and should probably only be given to a trusted
```

```
 780 group.
 781 .sp
 782 .LP
 783 When mounting a filesystem image, take care to use appropriate mount options.
 784 In particular, the \fBnosuid\fR mount option might be appropriate for \fBUFS\fR
 785 images whose origin is unknown. Also, some options might not be useful or
 786 appropriate, like \fBlogging\fR or \fBforcedirectio\fR for \fBUFS\fR. For
 787 compatibility purposes, a raw device is also exported along with the block
 788 device. For example, \fBnewfs\fR(1M) requires one.
 789 .sp
 790 .LP
 791 The output of \fBlofiadm\fR (without arguments) might change in future
 792 releases.
```