```
**********************************************************
   39127 Fri Apr 12 15:52:28 2013
new/usr/src/uts/common/io/blkdev/blkdev.c
*** NO COMMENTS ***
**********************************************************
______unchanged_portion_omitted_

1024 static int
1025 bd_ioctl(dev_t dev, int cmd, intptr_t arg, int flag, cred_t *credp, int *rvalp)
1026 {
1027         minor_t         inst;
1028         uint16_t        part;
1029         bd_t            *bd;
1030         void            *ptr = (void *)arg;
1031         int             rv;

1033         part = BDPART(dev);
1034         inst = BDINST(dev);

1036         if ((bd = ddi_get_soft_state(bd_state, inst)) == NULL) {
1037                 return (ENXIO);
1038         }

1040         rv = cmlb_ioctl(bd->d_cmlbh, dev, cmd, arg, flag, credp, rvalp, 0);
1041         if (rv != ENOTTY)
1042                 return (rv);

1044         switch (cmd) {
1045         case DKIOCGMEDIAINFO: {
1046                 struct dk_minfo minfo;

1048                 /* make sure our state information is current */
1049                 bd_update_state(bd);
1050                 bzero(&minfo, sizeof (minfo));
1051                 minfo.dki_media_type = DK_FIXED_DISK;
1052                 minfo.dki_lbsize = (1U << bd->d_blkshift);
1053                 minfo.dki_capacity = bd->d_numblks;
1054                 if (ddi_copyout(&minfo, ptr, sizeof (minfo), flag))  {
1055                         return (EFAULT);
1056                 }
1057                 return (0);
1058         }
1059         case DKIOCINFO: {
1060                 struct dk_cinfo cinfo;
1061                 bzero(&cinfo, sizeof (cinfo));
1062                 cinfo.dki_ctype = DKC_BLKDEV;
1063                 cinfo.dki_cnum = ddi_get_instance(ddi_get_parent(bd->d_dip));
1064                 (void) snprintf(cinfo.dki_cname, sizeof (cinfo.dki_cname),
1065                     "%s", ddi_driver_name(ddi_get_parent(bd->d_dip)));
1066                 (void) snprintf(cinfo.dki_dname, sizeof (cinfo.dki_dname),
1067                     "%s", ddi_driver_name(bd->d_dip));
1068                 cinfo.dki_unit = inst;
1069                 cinfo.dki_flags = DKI_FMTVOL;
1070                 cinfo.dki_partition = part;
1071                 cinfo.dki_maxtransfer = bd->d_maxxfer / DEV_BSIZE;
1072                 cinfo.dki_addr = 0;
1073                 cinfo.dki_slave = 0;
1074                 cinfo.dki_space = 0;
1075                 cinfo.dki_prio = 0;
1076                 cinfo.dki_vec = 0;
1077                 if (ddi_copyout(&cinfo, ptr, sizeof (cinfo), flag))  {
1078                         return (EFAULT);
1079                 }
1080                 return (0);
1081         }
1082         case DKIOCREMOVABLE: {
```



```
1083                 int i;
1084                 i = bd->d_removable ? 1 : 0;
1085                 if (ddi_copyout(&i, ptr, sizeof (i), flag)) {
1086                         return (EFAULT);
1087                 }
1088                 return (0);
1089         }
1090         case DKIOCHOTPLUGGABLE: {
1091                 int i;
1092                 i = bd->d_hotpluggable ? 1 : 0;
1093                 if (ddi_copyout(&i, ptr, sizeof (i), flag)) {
1094                         return (EFAULT);
1095                 }
1096                 return (0);
1097         }
1098         case DKIOCREADONLY: {
1099                 int i;
1100                 i = bd->d_rdonly ? 1 : 0;
1101                 if (ddi_copyout(&i, ptr, sizeof (i), flag)) {
1102                         return (EFAULT);
1103                 }
1104                 return (0);
1105         }
1106         case DKIOCSTATE: {
1107                 enum dkio_state state;
1108                 if (ddi_copyin(ptr, &state, sizeof (state), flag)) {
1109                         return (EFAULT);
1110                 }
1111                 if ((rv = bd_check_state(bd, &state)) != 0) {
1112                         return (rv);
1113                 }
1114                 if (ddi_copyout(&state, ptr, sizeof (state), flag)) {
1115                         return (EFAULT);
1116                 }
1117                 return (0);
1118         }
1119         case DKIOCFLUSHWRITECACHE: {
1120                 struct dk_callback *dkc = NULL;

1122                 if (flag & FKIOCTL)
1123                         dkc = (void *)arg;

1125                 rv = bd_flush_write_cache(bd, dkc);
1126                 return (rv);
1127         }

1129         default:
1130                 if (bd->d_ops.o_ioctl != NULL) {
1131                         rv = bd->d_ops.o_ioctl(dev, cmd, arg, flag, credp,
1132                             rvalp);
1133                 } else {
1134                         /* Unsupported ioctl ==> return ENOTTY. */
1135                         rv = ENOTTY;
1130                 break;

1136                 }
1137                 /* FALLTHRU */
1138         }
1139         return (rv);
1133         return (ENOTTY);
1140 }
______unchanged_portion_omitted_

1526 /*
1527  * Functions for device drivers.
1528  */
```

```
1529 bd_handle_t
1530 bd_alloc_handle(void *private, bd_ops_t *ops, ddi_dma_attr_t *dma, int kmflag)
1531 {
1532         bd_handle_t     hdl;

1534         hdl = kmem_zalloc(sizeof (*hdl), kmflag);
1535         if (hdl == NULL)
1536                 return (NULL);

1538         /*
1539          * Cheesy versioning handling.  We've only appended members into
1540          * bd_ops as we grew from v0 to v1.  Since we zalloc hdl, the
1541          * ioctl ops will be NULL anyway.  So for the old version, we
1542          * copy over only the v0 elements.
1543          */
1544         switch (ops->o_version) {
1545         case BD_OPS_VERSION_0:
1546                 /* Don't copy the last pointer in the structure. */
1547                 bcopy(ops, &hdl->h_ops, sizeof (*ops) - sizeof (void *));
1548                 break;
1549         case BD_OPS_VERSION_1:
1529         if (hdl != NULL) {
1550                 hdl->h_ops = *ops;
1551                 break;
1552         default:
1553                 kmem_free(hdl, sizeof (*hdl));
1554                 cmn_err(CE_WARN, "Unsupported blkdev ops version %d.\n",
1555                     ops->o_version);
1556                 return (NULL);
1557                 /* NOTREACHED */
1558         }
1559         hdl->h_dma = dma;
1560         hdl->h_private = private;
1533         }

1562         return (hdl);
1563 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4867 Fri Apr 12 15:52:29 2013
new/usr/src/uts/common/sys/blkdev.h
*** NO COMMENTS ***
**********************************************************
______unchanged_portion_omitted_

 121 #define BD_INFO_FLAG_REMOVABLE          (1U << 0)
 122 #define BD_INFO_FLAG_HOTPLUGGABLE       (1U << 1)
 123 #define BD_INFO_FLAG_READ_ONLY          (1U << 2)

 125 struct bd_ops {
 126         int     o_version;
 127         void    (*o_drive_info)(void *, bd_drive_t *);
 128         int     (*o_media_info)(void *, bd_media_t *);
 129         int     (*o_devid_init)(void *, dev_info_t *, ddi_devid_t *);
 130         int     (*o_sync_cache)(void *, bd_xfer_t *);
 131         int     (*o_read)(void *, bd_xfer_t *);
 132         int     (*o_write)(void *, bd_xfer_t *);
 133         int     (*o_ioctl)(dev_t, int, intptr_t, int, cred_t *, int *);
 134 };

 136 #define BD_OPS_VERSION_0                0
 137 #define BD_OPS_VERSION_1                1
 138 #define BD_OPS_VERSION                  1       /* Should be latest! */

 140 /*
 141  * Note, one handler *per* address.  Drivers with multiple targets at
 142  * different addresses must use separate handles.
 143  */
 144 bd_handle_t     bd_alloc_handle(void *, bd_ops_t *, ddi_dma_attr_t *, int);
 145 void            bd_free_handle(bd_handle_t);
 146 int             bd_attach_handle(dev_info_t *, bd_handle_t);
 147 int             bd_detach_handle(bd_handle_t);
 148 void            bd_state_change(bd_handle_t);
 149 void            bd_xfer_done(bd_xfer_t *, int);
 150 void            bd_mod_init(struct dev_ops *);
 151 void            bd_mod_fini(struct dev_ops *);

 153 #ifdef __cplusplus
 154 }
______unchanged_portion_omitted_
```