94

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/* defines and includes common among the fsck.jfs modules */
#include "xfsckint.h"

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * superblock buffer pointer
  *
  *      defined in xchkdsk.c
  */
extern struct superblock *sb_ptr;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * fsck aggregate info structure pointer
  *
  *      defined in xchkdsk.c
  */
extern struct fsck_agg_record *agg_recptr;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * fsck block map info structure pointer
  *
  *      defined in xchkdsk.c
  */
extern struct fsck_bmap_record *bmap_recptr;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * For message processing
  *
  *      defined in xchkdsk.c
  */

extern char message_parm_0[];
extern char message_parm_1[];
extern char message_parm_2[];
extern char message_parm_3[];
extern char message_parm_4[];
extern char message_parm_5[];
extern char message_parm_6[];
extern char message_parm_7[];
extern char message_parm_8[];
extern char message_parm_9[];

extern char *msgprms[];
extern int16_t msgprmidx[];

extern char *verbose_msg_ptr;

extern char *MsgText[];

extern char *Vol_Label;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * Device information.
  *
  *      defined in xchkdsk.c
  */
extern HFILE Dev_IOPort;
extern uint32_t Dev_blksize;

/* VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
 *
 * The following are internal to this file
 *
 */

struct fsck_stree_proc_parms {
        dmtree_t *buf_tree;
        int8_t *buf_stree;
        int8_t *wsp_stree;
```

```
91          int32_t nleafs;
92          int32_t l2nleafs;
93          int32_t leafidx;
94          int8_t budmin;
95          int page_level;
96          uint32_t page_ordno;
97          int8_t *lfval_error;
98          int8_t *intval_error;
99  };
100
101 int ctlpage_rebuild(int8_t);
102
103 int ctlpage_verify(int8_t);
104
105 int dmap_pwmap_rebuild(uint32_t *);
106
107 int dmap_pmap_verify(uint32_t *);
108
109 int dmap_tree_rebuild(int8_t *);
110
111 int dmap_tree_verify(int8_t *);
112
113 int dmappg_rebuild(int8_t *);
114
115 int dmappg_verify(int8_t *);
116
117 int init_bmap_info(void);
118
119 int Ln_tree_rebuild(int, int64_t, struct dmapctl **, int8_t *);
120
121 int Ln_tree_verify(int, int64_t, struct dmapctl **, int8_t *);
122
123 int stree_rebuild(struct fsck_stree_proc_parms *, int8_t *);
124
125 int stree_verify(struct fsck_stree_proc_parms *, int8_t *);
126
127 int verify_blkall_summary_msgs(void);
128
129 /* VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV */
130
131 /*****************************************************************************
132  * NAME: ctlpage_rebuild
133  *
134  * FUNCTION: Rebuild the control page of the filesystem block map.
135  *
136  * PARAMETERS:
137  *      max_buddy  − input − the data value in the root of the highest
138  *                          Lx page in the map.
139  *
140  * RETURNS:
141  *      success: FSCK_OK
142  *      failure: something else
143  */
144 int ctlpage_rebuild(int8_t max_buddy)
145 {
146         int bcr_rc = FSCK_OK;
147         int32_t highest_active_AG = 0, num_active_AGs = 0, num_inactive_AGs;
148         int64_t avg_free, actAG_free, inactAG_free;
149         int32_t l2nl, n, agidx, index, aglevel, agheight, agwidth, agstart;
150
151         bcr_rc = mapctl_get(bmap_recptr->bmpctl_agg_fsblk_offset,
152                             (void **) &(bmap_recptr->bmpctl_bufptr));
153
154         if (bcr_rc == FSCK_OK) {
155
156                 /* swap if on big endian machine */
157                 ujfs_swap_dbmap((struct dbmap *) bmap_recptr->bmpctl_bufptr);
158
159                 bmap_recptr->bmpctl_bufptr->dn_mapsize =
160                     bmap_recptr->total_blocks;
161                 bmap_recptr->bmpctl_bufptr->dn_nfree = bmap_recptr->free_blocks;
162                 bmap_recptr->bmpctl_bufptr->dn_l2nbperpage =
163                     agg_recptr->log2_blksperpg;
164                 bmap_recptr->bmpctl_bufptr->dn_numag = agg_recptr->num_ag;
165
166                 bmap_recptr->bmpctl_bufptr->dn_maxlevel =
167                     BMAPSZTOLEV(agg_recptr->sb_agg_fsblk_length);
168
169                 /* check out the active AGs */
170                 for (agidx = 0; (agidx < MAXAG); agidx++) {
171                         if (bmap_recptr->AGActive[agidx]) {
172                                 highest_active_AG = agidx;
173                                 num_active_AGs++;
174                         }
175                 }       /* end check out the active AGs */
176
177                 bmap_recptr->bmpctl_bufptr->dn_maxag = highest_active_AG;
178
179                 num_inactive_AGs = agg_recptr->num_ag − num_active_AGs;
180                 inactAG_free = num_inactive_AGs * sb_ptr->s_agsize;
```

```
                actAG_free = bmap_recptr->free_blocks − inactAG_free;
                avg_free = actAG_free / num_active_AGs;

                if ((bmap_recptr->bmpctl_bufptr->dn_agpref > highest_active_AG) || (bmap_recptr->bmpctl_bufptr->dn_agfree[bmap_recptr->bmpctl_bufptr->dn_agpref] < avg_free)) {
                        /* preferred AG is not valid */
                        if (avg_free == 0) {
                                bmap_recptr->bmpctl_bufptr->dn_agpref = 0;
                        } else {
                                bmap_recptr->bmpctl_bufptr->dn_agpref = −1;
                                for (agidx = 0; ((agidx < MAXAG) &&
                                                 (bmap_recptr->bmpctl_bufptr->
                                                  dn_agpref < 0)); agidx++) {
                                        if (bmap_recptr->AGFree_tbl[agidx] >= avg_free) {
                                                bmap_recptr->bmpctl_bufptr->
                                                    dn_agpref = agidx;
                                        }
                                }
                        }
                }
                aglevel = BMAPSZTOLEV(sb_ptr->s_agsize);
                l2nl =
                    agg_recptr->log2_blksperag − (L2BPERDMAP +
                                                  aglevel * L2LPERCTL);
                agheight = l2nl >> 1;
                agwidth = 1 << (l2nl − (agheight << 1));
                for (index = 5 − agheight, agstart = 0, n = 1; index > 0;
                     index−−) {
                        agstart += n;
                        n <<= 2;
                }

                bmap_recptr->bmpctl_bufptr->dn_aglevel = aglevel;
                bmap_recptr->bmpctl_bufptr->dn_agheigth = agheight;
                bmap_recptr->bmpctl_bufptr->dn_agwidth = agwidth;
                bmap_recptr->bmpctl_bufptr->dn_agstart = agstart;
                bmap_recptr->bmpctl_bufptr->dn_agl2size =
                    agg_recptr->log2_blksperag;
                bmap_recptr->bmpctl_bufptr->dn_agsize = sb_ptr->s_agsize;
                bmap_recptr->bmpctl_bufptr->dn_maxfreebud = max_buddy;

                for (agidx = 0; (agidx < MAXAG); agidx++) {
                        /* rebuild the free list */
                        bmap_recptr->bmpctl_bufptr->dn_agfree[agidx] =
                            bmap_recptr->AGFree_tbl[agidx];
                }

                /* swap if on big endian machine */
                ujfs_swap_dbmap(bmap_recptr->bmpctl_bufptr);

                /*
                 * write the updated control page back onto the device
                 */
                bcr_rc = mapctl_put((void *) bmap_recptr->bmpctl_bufptr);
        }
        return (bcr_rc);
}

/*****************************************************************************
 * NAME: ctlpage_verify
 *
 * FUNCTION: Verify the control page of the filesystem block map.
 *
 * PARAMETERS:
 *      max_buddy  − input − the data value which should be in the root
 *                          of the highest Lx page in the map.
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int ctlpage_verify(int8_t max_buddy)
{
        int bcv_rc = FSCK_OK;
        unsigned agidx;
        int32_t max_level = 0;
        int32_t highest_active_AG = 0;
        int32_t l2nl, n, index, aglevel, agheight, agwidth, agstart;

        bcv_rc = mapctl_get(bmap_recptr->bmpctl_agg_fsblk_offset,
                            (void **) &(bmap_recptr->bmpctl_bufptr));

        if (bcv_rc == FSCK_OK) {
                /* got the control page in the buffer */

                /* swap if on big endian machine */
                ujfs_swap_dbmap((struct dbmap *) bmap_recptr->bmpctl_bufptr);

                if (bmap_recptr->bmpctl_bufptr->dn_mapsize != bmap_recptr->total_blocks) {
                        /* bad number of blocks in the aggregate */
```

```
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCASB, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_nfree != bmap_recptr->free_blocks) {
                        /* bad number of free blocks in the aggregate */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCNF, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_l2nbperpage != agg_recptr->log2_blksperpg) {
                        /* bad log2( blocks per page ) */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCL2BPP, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_numag != agg_recptr->num_ag) {
                        /* bad number of alloc groups */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCNAG, 0);
                }
                max_level = BMAPSZTOLEV(agg_recptr->sb_agg_fsblk_length);
                if (bmap_recptr->bmpctl_bufptr->dn_maxlevel != max_level) {
                        /* bad maximum block map level */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCMXLVL, 0);
                }
                for (agidx = 0; (agidx < MAXAG); agidx++) {
                        /* check out the active AGs */
                        if (bmap_recptr->AGActive[agidx]) {
                                highest_active_AG = agidx;
                        }
                }
                /*
                 * format does not include blocks allocated to the bad block inode
                 * when it determines the dn_maxag.  Subsequent activity may or
                 * may not have altered dn_maxag based on blocks allocated to the
                 * bad block inode.  All we know for sure is that dn_maxag shouldn't
                 * be larger than appropriate for the highest allocated block.
                 */
                if (bmap_recptr->bmpctl_bufptr->dn_maxag > highest_active_AG) {
                        /* bad highest active alloc group */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCMAAG, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_agpref > highest_active_AG) {
                        /* bad preferred alloc group */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCPAG, 0);
                }
                aglevel = BMAPSZTOLEV(sb_ptr->s_agsize);
                l2nl =
                    agg_recptr->log2_blksperag - (L2BPERDMAP +
                                                  aglevel * L2LPERCTL);
                agheight = l2nl >> 1;
                agwidth = 1 << (l2nl - (agheight << 1));
                for (index = 5 - agheight, agstart = 0, n = 1; index > 0;
                     index--) {
                        agstart += n;
                        n <<= 2;
                }               /* end for index */

                if (bmap_recptr->bmpctl_bufptr->dn_aglevel != aglevel) {
                        /* bad level holding an AG */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCDMCLAG, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_agheigth != agheight) {
                        /* bad dmapctl height holding an AG */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCDMCLH, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_agwidth != agwidth) {
                        /* bad width at level holding an AG */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCDMCLW, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_agstart != agstart) {
                        /* bad start idx at level holding an AG */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCDMCSTI, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_agl2size != agg_recptr->log2_blksperag) {
                        /* bad log2(fsblks per AG) */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCL2BPAG, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_agsize != sb_ptr->s_agsize) {
                        /* bad fsblks per AG */
                        bmap_recptr->ctl_other_error = -1;
                        fsck_send_msg(fsck_BMAPCBPAG, 0);
                }
                if (bmap_recptr->bmpctl_bufptr->dn_maxfreebud != max_buddy) {
```

```
360                         /* bad max free buddy system */
361                         bmap_recptr->ctl_other_error = -1;
362                         fsck_send_msg(fsck_BMAPCBMXB, 0);
363                 }
364                 for (agidx = 0; (agidx < MAXAG); agidx++) {
365                         /* verify the free list */
366                         if (bmap_recptr->bmpctl_bufptr->dn_agfree[agidx] != bmap_recptr->AGFree_tbl[agidx]) {
367                                 /* bad free blocks in the AG */
368                                 bmap_recptr->ctl_fctl_error = -1;
369                                 sprintf(message_parm_0, "%d", agidx);
370                                 msgprms[0] = message_parm_0;
371                                 msgprmidx[0] = 0;
372                                 fsck_send_msg(fsck_BMAPCAGNF, 1);
373                         }
374                 }
375         }
376         return (bcv_rc);
377 }
378
379 /*****************************************************************************
380  * NAME: dmap_pwmap_rebuild
381  *
382  * FUNCTION:  Rebuild the pmap in the current block map dmap page.
383  *
384  * PARAMETERS:
385  *      pmap_freeblks  - input - pointer to a variable in which the number
386  *                               of free blocks described by the dmap page
387  *                               will be returned.
388  *
389  * RETURNS:
390  *      success: FSCK_OK
391  *      failure: something else
392  */
393 int dmap_pwmap_rebuild(uint32_t * pmap_freeblks)
394 {
395         int bdpr_rc = FSCK_OK;
396         uint32_t bitmask;
397         int64_t wsp_pagenum;
398         uint32_t wsp_byteoffset;
399         struct fsck_blk_map_page *wsp_page;
400         uint32_t *wsp_bits = NULL;
401         int32_t map_wordidx, word_bitidx;
402         int8_t max_buddy;
403
404         /*
405          * locate the section of the workspace bit map which corresponds
406          * to this dmap's pmap
407          */
408         bdpr_rc = blkmap_find_bit(bmap_recptr->dmap_1stblk, &wsp_pagenum,
409                                   &wsp_byteoffset, &bitmask);
410         if (bdpr_rc == FSCK_OK) {
411                 bdpr_rc = blkmap_get_page(wsp_pagenum, &wsp_page);
412                 if (bdpr_rc == FSCK_OK) {
413                         wsp_bits =
414                             (uint32_t *) ((char *) wsp_page + wsp_byteoffset);
415                 }
416         }
417
418         *pmap_freeblks = 0;
419
420         if (bdpr_rc == FSCK_OK) {
421                 for (map_wordidx = 0; (map_wordidx < LPERDMAP); map_wordidx++) {
422                         max_buddy =
423                             ujfs_maxbuddy((char *) &(wsp_bits[map_wordidx]));
424                         bmap_recptr->dmap_wsp_sleafs[map_wordidx] = max_buddy;
425
426                         bmap_recptr->dmap_bufptr->wmap[map_wordidx] =
427                             wsp_bits[map_wordidx];
428                         /*
429                          * copy the word from workspace to buffer,
430                          * into both the working map and the permanent map.
431                          */
432                         bmap_recptr->dmap_bufptr->pmap[map_wordidx] = wsp_bits[map_wordidx];
433                         /*
434                          * count the free blocks described by the word
435                          */
436                         bitmask = 0x80000000u;
437                         for (word_bitidx = 0; (word_bitidx < DBWORD);
438                              word_bitidx++) {
439
440                                 if (!(wsp_bits[map_wordidx] & bitmask)) {
441                                         /* it's free */
442                                         (*pmap_freeblks)++;
443                                         agg_recptr->free_blocks_in_aggregate++;
444                                 } else {
445                                         /* it's allocated */
446                                         agg_recptr->blocks_used_in_aggregate++;
447                                 }       /* end else it's allocated */
448
449                                 bitmask = bitmask >> 1; /* advance to the next bit */
```

```
                        }
                }
        }
        return (bdpr_rc);
}

/*****************************************************************************
 * NAME: dmap_pmap_verify
 *
 * FUNCTION:  Verify the pmap in the current block map dmap page.
 *
 * PARAMETERS:
 *      pmap_freeblks  − input − pointer to a variable in which the number
 *                              of free blocks described by the dmap page
 *                              will be returned.
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int dmap_pmap_verify(uint32_t * pmap_freeblks)
{
        int bdpv_rc = FSCK_OK;
        uint32_t bitmask;
        int64_t wsp_pagenum;
        uint32_t wsp_byteoffset;
        struct fsck_blk_map_page *wsp_page;
        uint32_t *wsp_bits = NULL;
        int32_t map_wordidx, word_bitidx;
        int8_t max_buddy;
        uint32_t unmarked_range_first_ordno = 0;
        int32_t unmarked_range_wordidx = 0;
        int32_t unmarked_range_bitidx = 0;
        int64_t size_of_unmarked_range = 0;
        uint32_t marked_range_first_ordno = 0;
        int32_t marked_range_wordidx = 0;
        int32_t marked_range_bitidx = 0;
        int64_t size_of_marked_range = 0;

        /*
         * locate the section of the workspace bit map which corresponds
         * to this dmap's pmap
         */
        bdpv_rc = blkmap_find_bit(bmap_recptr->dmap_1stblk, &wsp_pagenum,
                                  &wsp_byteoffset, &bitmask);
        if (bdpv_rc == FSCK_OK) {
                bdpv_rc = blkmap_get_page(wsp_pagenum, &wsp_page);
                if (bdpv_rc == FSCK_OK) {       /* got the page */
                        wsp_bits =
                            (uint32_t *) ((char *) wsp_page + wsp_byteoffset);
                }
        }

        *pmap_freeblks = 0;

        if (bdpv_rc == FSCK_OK) {
                for (map_wordidx = 0; (map_wordidx < LPERDMAP); map_wordidx++) {
                        max_buddy =
                            ujfs_maxbuddy((char *) &(wsp_bits[map_wordidx]));
                        bmap_recptr->dmap_wsp_sleafs[map_wordidx] = max_buddy;

                        bitmask = 0x80000000u;
                        for (word_bitidx = 0; (word_bitidx < DBWORD);
                             word_bitidx++) {

                                if (wsp_bits[map_wordidx] & bitmask) {
                                        agg_recptr->blocks_used_in_aggregate++;
                                        if (!(bmap_recptr->dmap_bufptr->pmap[map_wordidx] & bitmask)) {
                                                /* pmap says not */
                                                bmap_recptr->dmap_pmap_error =
                                                    −1;

                                                if (size_of_unmarked_range == 0) {

                                                        if (size_of_marked_range != 0) {
                                                                sprintf
                                                                    (message_parm_0,
                                                                     "%lld",
                                                                     (long long)
                                                                     size_of_marked_range);
                                                                msgprms[0] =
                                                                    message_parm_0;
                                                                msgprmidx[0] =
                                                                    0;
                                                                sprintf
                                                                    (message_parm_1,
                                                                     "%d",
                                                                     marked_range_first_ordno);
                                                                msgprms[1] =
                                                                    message_parm_1;
```

```
                                                msgprmidx[1] =
                                                    0;
                                                sprintf
                                                    (message_parm_2,
                                                     "%d",
                                                     marked_range_wordidx);
                                                msgprms[2] =
                                                    message_parm_2;
                                                msgprmidx[2] =
                                                    0;
                                                sprintf
                                                    (message_parm_3,
                                                     "%d",
                                                     marked_range_bitidx);
                                                msgprms[3] =
                                                    message_parm_3;
                                                msgprmidx[3] =
                                                    0;
                                                fsck_send_msg
                                                    (fsck_PMAPSBOFF,
                                                     4);
                                                size_of_marked_range
                                                    = 0;
                                        }
                                        unmarked_range_first_ordno
                                            =
                                            bmap_recptr->
                                            dmappg_ordno;
                                        unmarked_range_wordidx =
                                            map_wordidx;
                                        unmarked_range_bitidx =
                                            word_bitidx;
                                        size_of_unmarked_range =
                                            1;
                                } else {
                                        /* not the first in the range */
                                        size_of_unmarked_range++;
                                }
                        } else {
                                /* pmap agrees */
                                if (size_of_marked_range != 0) {
                                        /* marked range ended */
                                        sprintf(message_parm_0,
                                                "%lld",
                                                (long long)
                                                size_of_marked_range);
                                        msgprms[0] =
                                            message_parm_0;
                                        msgprmidx[0] = 0;
                                        sprintf(message_parm_1,
                                                "%d",
                                                marked_range_first_ordno);
                                        msgprms[1] =
                                            message_parm_1;
                                        msgprmidx[1] = 0;
                                        sprintf(message_parm_2,
                                                "%d",
                                                marked_range_wordidx);
                                        msgprms[2] =
                                            message_parm_2;
                                        msgprmidx[2] = 0;
                                        sprintf(message_parm_3,
                                                "%d",
                                                marked_range_bitidx);
                                        msgprms[3] =
                                            message_parm_3;
                                        msgprmidx[3] = 0;
                                        fsck_send_msg
                                            (fsck_PMAPSBOFF, 4);
                                        size_of_marked_range =
                                            0;
                                }
                                if (size_of_unmarked_range != 0) {
                                        /* unmarked range ended */
                                        sprintf(message_parm_0,
                                                "%lld",
                                                (long long)
                                                size_of_unmarked_range);
                                        msgprms[0] =
                                            message_parm_0;
                                        msgprmidx[0] = 0;
                                        sprintf(message_parm_1,
                                                "%d",
                                                unmarked_range_first_ordno);
                                        msgprmidx[1] = 0;
                                        msgprms[1] =
                                            message_parm_1;
                                        sprintf(message_parm_2,
                                                "%d",
                                                unmarked_range_wordidx);
```

```
                                        msgprmidx[2] = 0;
                                        msgprms[2] =
                                            message_parm_2;
                                        sprintf(message_parm_3,
                                                "%d",
                                                unmarked_range_bitidx);
                                        msgprmidx[3] = 0;
                                        msgprms[3] =
                                            message_parm_3;
                                        fsck_send_msg
                                            (fsck_PMAPSBON, 4);
                                        size_of_unmarked_range =
                                            0;
                                }
                        }
                } else {
                        /* fsck says not allocated */
                        (*pmap_freeblks)++;
                        agg_recptr->free_blocks_in_aggregate++;

                        if ((bmap_recptr->dmap_bufptr->pmap[map_wordidx] & bitmask)) {
                                /* pmap says yes */
                                bmap_recptr->dmap_pmap_error =
                                    -1;

                                if (size_of_marked_range == 0) {
                                        if (size_of_unmarked_range != 0) {
                                                sprintf
                                                    (message_parm_0,
                                                     "%lld",
                                                     (long long)
                                                     size_of_unmarked_range);
                                                msgprms[0] =
                                                    message_parm_0;
                                                msgprmidx[0] =
                                                    0;
                                                sprintf
                                                    (message_parm_1,
                                                     "%d",
                                                     unmarked_range_first_ordno);
                                                msgprms[1] =
                                                    message_parm_1;
                                                msgprmidx[1] =
                                                    0;
                                                sprintf
                                                    (message_parm_2,
                                                     "%d",
                                                     unmarked_range_wordidx);
                                                msgprms[2] =
                                                    message_parm_2;
                                                msgprmidx[2] =
                                                    0;
                                                sprintf
                                                    (message_parm_3,
                                                     "%d",
                                                     unmarked_range_bitidx);
                                                msgprms[3] =
                                                    message_parm_3;
                                                msgprmidx[3] =
                                                    0;
                                                fsck_send_msg
                                                    (fsck_PMAPSBON,
                                                     4);
                                                size_of_unmarked_range
                                                    = 0;
                                        }
                                        marked_range_first_ordno
                                            =
                                            bmap_recptr->
                                            dmappg_ordno;
                                        marked_range_wordidx =
                                            map_wordidx;
                                        marked_range_bitidx =
                                            word_bitidx;
                                        size_of_marked_range =
                                            1;
                                } else {
                                        /* not the first in the range */
                                        size_of_marked_range++;
                                }
                        } else {
                                /* pmap agrees */
                                if (size_of_marked_range != 0) {
                                        sprintf(message_parm_0,
                                                "%lld",
                                                (long long)
                                                size_of_marked_range);
                                        msgprms[0] =
                                            message_parm_0;
                                        msgprmidx[0] = 0;
```

```
                                                sprintf(message_parm_1,
                                                        "%d",
                                                        marked_range_first_ordno);
                                                msgprms[1] =
                                                    message_parm_1;
                                                msgprmidx[1] = 0;
                                                sprintf(message_parm_2,
                                                        "%d",
                                                        marked_range_wordidx);
                                                msgprms[2] =
                                                    message_parm_2;
                                                msgprmidx[2] = 0;
                                                sprintf(message_parm_3,
                                                        "%d",
                                                        marked_range_bitidx);
                                                msgprms[3] =
                                                    message_parm_3;
                                                msgprmidx[3] = 0;
                                                fsck_send_msg
                                                    (fsck_PMAPSBOFF, 4);
                                                size_of_marked_range =
                                                    0;
                                        }
                                        if (size_of_unmarked_range != 0) {
                                                /* unmarked range ended */
                                                sprintf(message_parm_0,
                                                        "%lld",
                                                        (long long)
                                                        size_of_unmarked_range);
                                                msgprms[0] =
                                                    message_parm_0;
                                                msgprmidx[0] = 0;
                                                sprintf(message_parm_1,
                                                        "%d",
                                                        unmarked_range_first_ordno);
                                                msgprms[1] =
                                                    message_parm_1;
                                                msgprmidx[1] = 0;
                                                sprintf(message_parm_2,
                                                        "%d",
                                                        unmarked_range_wordidx);
                                                msgprms[2] =
                                                    message_parm_2;
                                                msgprmidx[2] = 0;
                                                sprintf(message_parm_3,
                                                        "%d",
                                                        unmarked_range_bitidx);
                                                msgprms[3] =
                                                    message_parm_3;
                                                msgprmidx[3] = 0;
                                                fsck_send_msg
                                                    (fsck_PMAPSBON, 4);
                                                size_of_unmarked_range =
                                                    0;
                                        }
                                }
                        }

                        /* advance to the next bit */
                        bitmask = bitmask >> 1;
                }
        }

        if (size_of_marked_range != 0) {
                /* marked range ended */
                sprintf(message_parm_0, "%lld",
                        (long long) size_of_marked_range);
                msgprms[0] = message_parm_0;
                msgprmidx[0] = 0;
                sprintf(message_parm_1, "%d", marked_range_first_ordno);
                msgprms[1] = message_parm_1;
                msgprmidx[1] = 0;
                sprintf(message_parm_2, "%d", marked_range_wordidx);
                msgprms[2] = message_parm_2;
                msgprmidx[2] = 0;
                sprintf(message_parm_3, "%d", marked_range_bitidx);
                msgprms[3] = message_parm_3;
                msgprmidx[3] = 0;
                fsck_send_msg(fsck_PMAPSBOFF, 4);
                size_of_marked_range = 0;
        }
        if (size_of_unmarked_range != 0) {
                /* unmarked range ended */
                sprintf(message_parm_0, "%lld",
                        (long long) size_of_unmarked_range);
                msgprms[0] = message_parm_0;
                msgprmidx[0] = 0;
                sprintf(message_parm_1, "%d",
                        unmarked_range_first_ordno);
                msgprms[1] = message_parm_1;
```

```
                        msgprmidx[1] = 0;
                        sprintf(message_parm_2, "%d", unmarked_range_wordidx);
                        msgprms[2] = message_parm_2;
                        msgprmidx[2] = 0;
                        sprintf(message_parm_3, "%d", unmarked_range_bitidx);
                        msgprms[3] = message_parm_3;
                        msgprmidx[3] = 0;
                        fsck_send_msg(fsck_PMAPSBON, 4);
                        size_of_unmarked_range = 0;
                }
        }
        return (bdpv_rc);
}

/*****************************************************************************
 * NAME: dmap_tree_rebuild
 *
 * FUNCTION:  Rebuild the tree in the current block map dmap page.
 *
 * PARAMETERS:
 *      root_data  − input − pointer to a variable in which the data value
 *                          stored in the root of the tree will be returned.
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int dmap_tree_rebuild(int8_t * root_data)
{
        int bdsr_rc = FSCK_OK;
        struct fsck_stree_proc_parms stree_proc_parms;
        struct fsck_stree_proc_parms *prms_ptr;

        prms_ptr = &stree_proc_parms;

#define ppbt    prms_ptr−>buf_tree

        ppbt = (dmtree_t *) & (bmap_recptr−>dmap_bufptr−>tree);

        ppbt->dmt_nleafs = prms_ptr−>nleafs = LPERDMAP;
        ppbt->dmt_l2nleafs = prms_ptr−>l2nleafs = L2LPERDMAP;
        ppbt->dmt_leafidx = prms_ptr−>leafidx = LEAFIND;
        ppbt->dmt_height = 4;
        ppbt->dmt_budmin = prms_ptr−>budmin = BUDMIN;

        prms_ptr−>buf_stree = &(ppbt->dmt_stree[0]);
        prms_ptr−>wsp_stree = bmap_recptr−>dmap_wsp_stree;

        bdsr_rc = stree_rebuild(prms_ptr, root_data);

        return (bdsr_rc);
}

/*****************************************************************************
 * NAME: dmap_tree_verify
 *
 * FUNCTION:  Verify the tree in the current block map dmap page.
 *
 * PARAMETERS:
 *      root_data  − input − pointer to a variable in which the data value
 *                          which should be stored in the root of the tree
 *                          will be returned.
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int dmap_tree_verify(int8_t * root_data)
{
        int bdsv_rc = FSCK_OK;
        int8_t tree_spec_errors = 0;
        struct fsck_stree_proc_parms stree_proc_parms;
        struct fsck_stree_proc_parms *prms_ptr;

        prms_ptr = &stree_proc_parms;

        prms_ptr->buf_tree = (dmtree_t *) & (bmap_recptr->dmap_bufptr->tree);

        if (prms_ptr->buf_tree->dmt_nleafs != LPERDMAP) {
                /* wrong number of leafs */
                bmap_recptr->dmap_other_error = −1;
                tree_spec_errors = −1;
                msgprms[0] = message_parm_0;
                msgprmidx[0] = fsck_dmap;
                sprintf(message_parm_1, "%d", bmap_recptr−>dmappg_ordno);
                msgprms[1] = message_parm_1;
                msgprmidx[1] = 0;
                fsck_send_msg(fsck_BMAPBADNLF, 2);
        }
        if (prms_ptr−>buf_tree−>dmt_l2nleafs != L2LPERDMAP) {
```

```
900                     /* wrong log2(nleafs) */
901                     bmap_recptr->dmap_other_error = -1;
902                     tree_spec_errors = -1;
903                     msgprms[0] = message_parm_0;
904                     msgprmidx[0] = fsck_dmap;
905                     sprintf(message_parm_1, "%d", bmap_recptr->dmappg_ordno);
906                     msgprms[1] = message_parm_1;
907                     msgprmidx[1] = 0;
908                     fsck_send_msg(fsck_BMAPBADL2NLF, 2);
909             }
910             if (prms_ptr->buf_tree->dmt_leafidx != LEAFIND) {
911                     /* wrong 1st leaf index */
912                     bmap_recptr->dmap_other_error = -1;
913                     tree_spec_errors = -1;
914                     msgprms[0] = message_parm_0;
915                     msgprmidx[0] = fsck_dmap;
916                     sprintf(message_parm_1, "%d", bmap_recptr->dmappg_ordno);
917                     msgprms[1] = message_parm_1;
918                     msgprmidx[1] = 0;
919                     fsck_send_msg(fsck_BMAPBADLFI, 2);
920             }
921             if (prms_ptr->buf_tree->dmt_height != 4) {
922                     /* wrong stree height */
923                     bmap_recptr->dmap_other_error = -1;
924                     tree_spec_errors = -1;
925                     msgprms[0] = message_parm_0;
926                     msgprmidx[0] = fsck_dmap;
927                     sprintf(message_parm_1, "%d", bmap_recptr->dmappg_ordno);
928                     msgprms[1] = message_parm_1;
929                     msgprmidx[1] = 0;
930                     fsck_send_msg(fsck_BMAPBADHT, 2);
931             }
932             if (prms_ptr->buf_tree->dmt_budmin != BUDMIN) {
933                     /* wrong min buddy value */
934                     bmap_recptr->dmap_other_error = -1;
935                     tree_spec_errors = -1;
936                     msgprms[0] = message_parm_0;
937                     msgprmidx[1] = fsck_dmap;
938                     sprintf(message_parm_1, "%d", bmap_recptr->dmappg_ordno);
939                     msgprms[1] = message_parm_1;
940                     msgprmidx[2] = 0;
941                     fsck_send_msg(fsck_BMAPBADBMN, 2);
942             }
943
944             /*
945              * if we found errors in the fields which specify the summary
946              * tree then we won't take the time to verify the tree itself.
947              *
948              * (The errors already detected would corrupt the summary tree,
949              *  so info about the bad tree would only be noise at this point.)
950              */
951             if (!tree_spec_errors) {
952                     /* tree specification fields are ok */
953                     prms_ptr->buf_stree = &(prms_ptr->buf_tree->dmt_stree[0]);
954                     prms_ptr->wsp_stree = bmap_recptr->dmap_wsp_stree;
955                     prms_ptr->nleafs = LPERDMAP;
956                     prms_ptr->l2nleafs = L2LPERDMAP;
957                     prms_ptr->leafidx = LEAFIND;
958                     prms_ptr->budmin = BUDMIN;
959                     prms_ptr->page_level = fsck_dmap;
960                     prms_ptr->page_ordno = bmap_recptr->dmappg_ordno;
961                     prms_ptr->lfval_error = &(bmap_recptr->dmap_slfv_error);
962                     prms_ptr->intval_error = &(bmap_recptr->dmap_slnv_error);
963
964                     bdsv_rc = stree_verify(prms_ptr, root_data);
965             }
966             return (bdsv_rc);
967     }
968
969     /*****************************************************************************
970      * NAME: dmappg_rebuild
971      *
972      * FUNCTION: Rebuild the current dmap page.
973      *
974      * PARAMETERS:
975      *      stree_root_data  - input - pointer to a variable in which to return
976      *                                 the the data value in the root of the
977      *                                 tree of the rebuilt page.
978      *
979      * RETURNS:
980      *      success: FSCK_OK
981      *      failure: something else
982      */
983     int dmappg_rebuild(int8_t * stree_root_data)
984     {
985             int bdmpr_rc = FSCK_OK;
986             uint32_t nblocks;
987             uint32_t dmap_freeblks;
988             uint32_t ag_num;
989
```

```
990             /*
991              * get the page to verify into the I/O buffer
992              */
993             bdmpr_rc =
994                 blktbl_dmap_get(bmap_recptr->dmap_1stblk,
995                                 &(bmap_recptr->dmap_bufptr));
996
997             if (bdmpr_rc == FSCK_OK) {
998                     /* the page is in the buffer */
999                     bmap_recptr->dmap_bufptr->start = bmap_recptr->dmap_1stblk;
1000                    nblocks =
1001                        MIN(BPERDMAP,
1002                            (bmap_recptr->total_blocks - bmap_recptr->dmap_1stblk));
1003                    bmap_recptr->dmap_bufptr->nblocks = nblocks;
1004                    bdmpr_rc = dmap_pwmap_rebuild(&dmap_freeblks);
1005                    /*
1006                     * subtract out any blocks which don't really exist but are
1007                     * described by a dmap (phantom blocks always appear to be in use)
1008                     */
1009                    agg_recptr->blocks_used_in_aggregate = agg_recptr->blocks_used_in_aggregate - (BPERDMAP - nblocks
    );
1010                    if (bdmpr_rc == FSCK_OK) {
1011                            /* nothing strange during pmap rebld */
1012                            bmap_recptr->free_blocks += dmap_freeblks;
1013                            ag_num =
1014                                bmap_recptr->dmap_1stblk >> agg_recptr->
1015                                log2_blksperag;
1016                            bmap_recptr->AGFree_tbl[ag_num] += dmap_freeblks;
1017
1018                            if (dmap_freeblks != nblocks) {
1019                                    /* not all of them are free */
1020                                    bmap_recptr->AGActive[ag_num] = -1;
1021                            }
1022                            bmap_recptr->dmap_bufptr->nfree = dmap_freeblks;
1023                            bdmpr_rc = dmap_tree_rebuild(stree_root_data);
1024                            /*
1025                             * write the page back to the device
1026                             */
1027                            if (bdmpr_rc == FSCK_OK) {
1028                                    bdmpr_rc =
1029                                        blktbl_dmap_put(bmap_recptr->dmap_bufptr);
1030                            }
1031                    }
1032            }
1033            return (bdmpr_rc);
1034    }
1035
1036    /*****************************************************************************
1037     * NAME: dmappg_verify
1038     *
1039     * FUNCTION: Validate the current dmap page.
1040     *
1041     * PARAMETERS:
1042     *      stree_root_data  - input - pointer to a variable in which to return
1043     *                                 the the data value which should be stored
1044     *                                 in the root of the tree of the page being
1045     *                                 validated (and may also be the one stored
1046     *                                 in the root of that tree)
1047     *
1048     * RETURNS:
1049     *      success: FSCK_OK
1050     *      failure: something else
1051     */
1052    int dmappg_verify(int8_t * stree_root_data)
1053    {
1054            int bdmpv_rc = FSCK_OK;
1055            uint32_t nblocks;
1056            uint32_t dmap_freeblks;
1057            uint32_t ag_num;
1058
1059            /*
1060             * get the page to verify into the I/O buffer
1061             */
1062            bdmpv_rc =
1063                blktbl_dmap_get(bmap_recptr->dmap_1stblk,
1064                                &(bmap_recptr->dmap_bufptr));
1065
1066            if (bdmpv_rc == FSCK_OK) {
1067                    /* the page is in the buffer */
1068                    if (bmap_recptr->dmap_bufptr->start != bmap_recptr->dmap_1stblk) {
1069                            /* bad starting block */
1070                            bmap_recptr->dmap_other_error = -1;
1071                            sprintf(message_parm_0, "%d",
1072                                    bmap_recptr->dmappg_ordno);
1073                            msgprms[0] = message_parm_0;
1074                            msgprmidx[0] = 0;
1075                            fsck_send_msg(fsck_DMAPBADSTRT, 1);
1076                    }
1077                    nblocks =
1078                        MIN(BPERDMAP,
```

```
1079                    (bmap_recptr->total_blocks − bmap_recptr->dmap_1stblk));
1080            if (bmap_recptr->dmap_bufptr->nblocks != nblocks) {
1081                    /* bad number of blocks */
1082                    bmap_recptr->dmap_other_error = −1;
1083                    sprintf(message_parm_0, "%d",
1084                            bmap_recptr->dmappg_ordno);
1085                    msgprms[0] = message_parm_0;
1086                    msgprmidx[0] = 0;
1087                    fsck_send_msg(fsck_DMAPBADNBLK, 1);
1088            }
1089            bdmpv_rc = dmap_pmap_verify(&dmap_freeblks);
1090            /*
1091             * subtract out any blocks which don't really exist but are described by a dmap
1092             * (phantom blocks always appear to be in use)
1093             */
1094            agg_recptr->blocks_used_in_aggregate = agg_recptr->blocks_used_in_aggregate − (BPERDMAP − nblocks
    );
1095            if (bdmpv_rc == FSCK_OK) {
1096                    /* nothing strange during pmap ver */
1097                    bmap_recptr->free_blocks += dmap_freeblks;
1098                    ag_num =
1099                        bmap_recptr->dmap_1stblk >> agg_recptr->
1100                        log2_blksperag;
1101                    bmap_recptr->AGFree_tbl[ag_num] += dmap_freeblks;
1102                    if (dmap_freeblks != nblocks) {
1103                            /* not all of them are free */
1104                            bmap_recptr->AGActive[ag_num] = −1;
1105                    }
1106                    if (bmap_recptr->dmap_bufptr->nfree != dmap_freeblks) {
1107                            /* bad number of free blocks */
1108                            bmap_recptr->dmap_other_error = −1;
1109                            sprintf(message_parm_0, "%d",
1110                                    bmap_recptr->dmappg_ordno);
1111                            msgprms[0] = message_parm_0;
1112                            msgprmidx[0] = 0;
1113                            fsck_send_msg(fsck_DMAPBADNFREE, 1);
1114                    }
1115                    bdmpv_rc = dmap_tree_verify(stree_root_data);
1116            }
1117    }
1118    return (bdmpv_rc);
1119 }
1120
1121 /*****************************************************************************
1122  * NAME: init_bmap_info
1123  *
1124  * FUNCTION:  Initialize the bmap fsck global data area, pointed to by
1125  *            bmap_recptr, used to control JFS bmap processing.
1126  *
1127  * PARAMETERS:  none
1128  *
1129  * RETURNS:
1130  *      success: FSCK_OK
1131  *      failure: something else
1132  */
1133 int init_bmap_info()
1134 {
1135         int bii_rc = FSCK_OK;
1136         unsigned agidx;
1137
1138         memset(bmap_recptr, 0, sizeof (struct fsck_bmap_record));
1139
1140         memcpy((void *) &(bmap_recptr->eyecatcher), (void *) "bmaprecd", 8);
1141         memcpy((void *) &(bmap_recptr->bmpctlinf_eyecatcher),
1142                (void *) "bmap ctl", 8);
1143         memcpy((void *) &(bmap_recptr->AGinf_eyecatcher), (void *) "AG info ",
1144                8);
1145         memcpy((void *) &(bmap_recptr->dmapinf_eyecatcher), (void *) "dmapinfo",
1146                8);
1147         memcpy((void *) &(bmap_recptr->L0inf_eyecatcher), (void *) "L0 info ",
1148                8);
1149         memcpy((void *) &(bmap_recptr->L1inf_eyecatcher), (void *) "L1 info ",
1150                8);
1151         memcpy((void *) &(bmap_recptr->L2inf_eyecatcher), (void *) "L2 info ",
1152                8);
1153
1154         bmap_recptr->bmpctl_bufptr =
1155             (struct dbmap *) agg_recptr->mapctl_buf_ptr;
1156         bmap_recptr->bmpctl_agg_fsblk_offset = BMAP_OFF / sb_ptr->s_bsize;
1157
1158         bmap_recptr->AGFree_tbl = &(agg_recptr->blkmap_wsp.AG_free[0]);
1159         bmap_recptr->dmap_wsp_stree =
1160             &(agg_recptr->blkmap_wsp.dmap_wsp_tree[0]);
1161         bmap_recptr->dmap_wsp_sleafs =
1162             &(agg_recptr->blkmap_wsp.dmap_wsp_leafs[0]);
1163         bmap_recptr->L0_wsp_stree = &(agg_recptr->blkmap_wsp.L0_wsp_tree[0]);
1164         bmap_recptr->L0_wsp_sleafs = &(agg_recptr->blkmap_wsp.L0_wsp_leafs[0]);
1165         bmap_recptr->L0_bufptr =
1166             (struct dmapctl *) &(agg_recptr->bmaplv_buf_ptr);
1167         bmap_recptr->L1_wsp_stree = &(agg_recptr->blkmap_wsp.L1_wsp_tree[0]);
```

```
1168          bmap_recptr->L1_wsp_sleafs = &(agg_recptr->blkmap_wsp.L1_wsp_leafs[0]);
1169          bmap_recptr->L1_bufptr =
1170              (struct dmapctl *) &(agg_recptr->bmaplv_buf_ptr);
1171          bmap_recptr->L2_wsp_stree = &(agg_recptr->blkmap_wsp.L2_wsp_tree[0]);
1172          bmap_recptr->L2_wsp_sleafs = &(agg_recptr->blkmap_wsp.L2_wsp_leafs[0]);
1173          bmap_recptr->L2_bufptr =
1174              (struct dmapctl *) &(agg_recptr->bmaplv_buf_ptr);
1175
1176          bmap_recptr->total_blocks =
1177              sb_ptr->s_size * sb_ptr->s_pbsize / sb_ptr->s_bsize;
1178          bmap_recptr->dmappg_count =
1179              (bmap_recptr->total_blocks + BPERDMAP - 1) / BPERDMAP;
1180          bmap_recptr->L0pg_count =
1181              (bmap_recptr->dmappg_count + LPERCTL - 1) / LPERCTL;
1182          if (bmap_recptr->L0pg_count > 1) {
1183                  bmap_recptr->L1pg_count =
1184                      (bmap_recptr->L0pg_count + LPERCTL - 1) / LPERCTL;
1185                  if (bmap_recptr->L1pg_count > 1) {
1186                          bmap_recptr->L2pg_count =
1187                              (bmap_recptr->L1pg_count + LPERCTL - 1) / LPERCTL;
1188                  } else {
1189                          bmap_recptr->L2pg_count = 0;
1190                  }
1191          } else {
1192                  bmap_recptr->L1pg_count = 0;
1193          }
1194
1195          bmap_recptr->free_blocks = 0;
1196          bmap_recptr->ctl_fctl_error = 0;
1197          bmap_recptr->ctl_other_error = 0;
1198
1199          for (agidx = 0; (agidx < MAXAG); agidx++) {
1200                  bmap_recptr->AGActive[agidx] = 0;
1201          }
1202
1203          bmap_recptr->dmappg_ordno = bmap_recptr->L0pg_ordno = 0;
1204          bmap_recptr->L1pg_ordno = bmap_recptr->L2pg_1stblk = 0;
1205
1206          bmap_recptr->dmappg_idx = bmap_recptr->L0pg_idx = 0;
1207          bmap_recptr->L1pg_idx = 0;
1208
1209          bmap_recptr->dmap_1stblk = bmap_recptr->L0pg_1stblk = 0;
1210          bmap_recptr->L1pg_1stblk = bmap_recptr->L2pg_1stblk = 0;
1211
1212          bmap_recptr->dmap_pmap_error = 0;
1213
1214          bmap_recptr->dmap_slfv_error = bmap_recptr->L0pg_slfv_error = 0;
1215          bmap_recptr->L1pg_slfv_error = bmap_recptr->L2pg_slfv_error = 0;
1216
1217          bmap_recptr->dmap_slnv_error = bmap_recptr->L0pg_slnv_error = 0;
1218          bmap_recptr->L1pg_slnv_error = bmap_recptr->L2pg_slnv_error = 0;
1219
1220          bmap_recptr->dmap_other_error = bmap_recptr->L0pg_other_error = 0;
1221          bmap_recptr->L1pg_other_error = bmap_recptr->L2pg_other_error = 0;
1222
1223          return (bii_rc);
1224  }
1225
1226  /*****************************************************************************
1227   * NAME: Ln_tree_rebuild
1228   *
1229   * FUNCTION:  Rebuild the tree in the current summary page.
1230   *
1231   * PARAMETERS:
1232   *      level         - input - Summary level of the bmap summary page
1233   *                              containing the tree to rebuild
1234   *      first_blk     - input - First aggregate block described by the bmap
1235   *                              summary page containing the tree to rebuild
1236   *      addr_buf_ptr  - input - pointer to a variable in which to return the
1237   *                              address of the buffer in which the page has
1238   *                              been rebuilt (and then written to the
1239   *                              aggregate)
1240   *      root_data     - input - pointer to a variable in which to return the
1241   *                              data value in the root of the rebuilt tree.
1242   *
1243   * RETURNS:
1244   *      success: FSCK_OK
1245   *      failure: something else
1246   */
1247  int Ln_tree_rebuild(int level, int64_t first_blk, struct dmapctl **addr_buf_ptr,
1248                      int8_t * root_data)
1249  {
1250          int blsr_rc = FSCK_OK;
1251          struct fsck_stree_proc_parms stree_proc_parms;
1252          struct fsck_stree_proc_parms *prms_ptr;
1253
1254          /*
1255           * get the page to verify into the I/O buffer
1256           */
1257          blsr_rc = blktbl_Ln_page_get(level, first_blk, addr_buf_ptr);
```

```
1258
1259            if (blsr_rc == FSCK_OK) {
1260                    /* the page is in the buffer */
1261                    prms_ptr = &stree_proc_parms;
1262                    /* these are just big trees */
1263                    prms_ptr->buf_tree = (dmtree_t *) * addr_buf_ptr;
1264
1265                    switch (level) {
1266                    case 0:{
1267                                    prms_ptr->wsp_stree = bmap_recptr->L0_wsp_stree;
1268                                    break;
1269                            }
1270                    case 1:{
1271                                    prms_ptr->wsp_stree = bmap_recptr->L1_wsp_stree;
1272                                    break;
1273                            }
1274                    default:{
1275                                    prms_ptr->wsp_stree = bmap_recptr->L2_wsp_stree;
1276                                    break;
1277                            }
1278                    }
1279
1280                    prms_ptr->buf_tree->dmt_nleafs = prms_ptr->nleafs = LPERCTL;
1281                    prms_ptr->buf_tree->dmt_l2nleafs = prms_ptr->l2nleafs =
1282                        L2LPERCTL;
1283                    prms_ptr->buf_tree->dmt_leafidx = prms_ptr->leafidx =
1284                        CTLLEAFIND;
1285                    prms_ptr->buf_tree->dmt_height = 5;
1286                    prms_ptr->buf_tree->dmt_budmin = L2BPERDMAP + level * L2LPERCTL;
1287                    prms_ptr->budmin = prms_ptr->buf_tree->dmt_budmin;
1288
1289                    prms_ptr->buf_stree = &(prms_ptr->buf_tree->dmt_stree[0]);
1290
1291                    blsr_rc = stree_rebuild(prms_ptr, root_data);
1292
1293                    /*
1294                     * put the page back into the file
1295                     */
1296                    if (blsr_rc == FSCK_OK) {
1297                            blsr_rc = blktbl_Ln_page_put(*addr_buf_ptr);
1298                    }
1299            }
1300            return (blsr_rc);
1301    }
1302
1303    /*****************************************************************************
1304     * NAME: Ln_tree_verify
1305     *
1306     * FUNCTION:  Verify the tree in the current summary page.
1307     *
1308     * PARAMETERS:
1309     *      level         - input - Summary level of the bmap summary page
1310     *                              containing the tree to verify
1311     *      first_blk     - input - First aggregate block described by the bmap
1312     *                              summary page containing the tree to verify
1313     *      addr_buf_ptr  - input - pointer to a variable in which to return the
1314     *                              address of the buffer into which the page has
1315     *                              been read so that its contents can be verified
1316     *      root_data     - input - pointer to a variable in which to return the
1317     *                              data value which should be in the root of the
1318     *                              tree....and may actually be stored there.
1319     *
1320     * RETURNS:
1321     *      success: FSCK_OK
1322     *      failure: something else
1323     */
1324    int Ln_tree_verify(int level, int64_t first_blk, struct dmapctl **addr_buf_ptr,
1325                       int8_t * root_data)
1326    {
1327            int blsv_rc = FSCK_OK;
1328            int8_t *other_errors;
1329            int8_t tree_spec_errors = 0;
1330            struct fsck_stree_proc_parms stree_proc_parms;
1331            struct fsck_stree_proc_parms *prms_ptr;
1332
1333            /*
1334             * get the page to verify into the I/O buffer
1335             */
1336            blsv_rc = blktbl_Ln_page_get(level, first_blk, addr_buf_ptr);
1337
1338            if (blsv_rc == FSCK_OK) {
1339                    /* the page is in the buffer */
1340                    prms_ptr = &stree_proc_parms;
1341                    /* these are just big trees */
1342                    prms_ptr->buf_tree = (dmtree_t *) * addr_buf_ptr;
1343
1344                    switch (level) {
1345                    case 0:{
1346                                    prms_ptr->wsp_stree = bmap_recptr->L0_wsp_stree;
1347                                    prms_ptr->page_ordno = bmap_recptr->L0pg_ordno;
```

```
1348                                    prms_ptr->lfval_error =
1349                                        &(bmap_recptr->L0pg_slfv_error);
1350                                    prms_ptr->intval_error =
1351                                        &(bmap_recptr->L0pg_slnv_error);
1352                                    prms_ptr->page_level = fsck_L0;
1353                                    other_errors = &(bmap_recptr->L0pg_other_error);
1354                                    break;
1355                            }
1356                    case 1:{
1357                                    prms_ptr->wsp_stree = bmap_recptr->L1_wsp_stree;
1358                                    prms_ptr->page_ordno = bmap_recptr->L1pg_ordno;
1359                                    prms_ptr->lfval_error =
1360                                        &(bmap_recptr->L1pg_slfv_error);
1361                                    prms_ptr->intval_error =
1362                                        &(bmap_recptr->L1pg_slnv_error);
1363                                    prms_ptr->page_level = fsck_L1;
1364                                    other_errors = &(bmap_recptr->L1pg_other_error);
1365                                    break;
1366                            }
1367                    default:{
1368                                    prms_ptr->wsp_stree = bmap_recptr->L2_wsp_stree;
1369                                    prms_ptr->page_ordno = 0;
1370                                    prms_ptr->lfval_error =
1371                                        &(bmap_recptr->L2pg_slfv_error);
1372                                    prms_ptr->intval_error =
1373                                        &(bmap_recptr->L2pg_slnv_error);
1374                                    prms_ptr->page_level = fsck_L2;
1375                                    other_errors = &(bmap_recptr->L2pg_other_error);
1376                                    break;
1377                            }
1378                    }
1379
1380                    if (prms_ptr->buf_tree->dmt_nleafs != LPERCTL) {
1381                            /* wrong number of leafs */
1382                            *other_errors = -1;
1383                            tree_spec_errors = -1;
1384                            msgprms[0] = message_parm_0;
1385                            msgprmidx[0] = prms_ptr->page_level;
1386                            sprintf(message_parm_1, "%d", prms_ptr->page_ordno);
1387                            msgprms[1] = message_parm_1;
1388                            msgprmidx[1] = 0;
1389                            fsck_send_msg(fsck_BMAPBADNLF, 2);
1390                    }
1391                    if (prms_ptr->buf_tree->dmt_l2nleafs != L2LPERCTL) {
1392                            /* wrong log2(nleafs) */
1393                            *other_errors = -1;
1394                            tree_spec_errors = -1;
1395                            msgprms[0] = message_parm_0;
1396                            msgprmidx[0] = prms_ptr->page_level;
1397                            sprintf(message_parm_1, "%d", prms_ptr->page_ordno);
1398                            msgprms[1] = message_parm_1;
1399                            msgprmidx[1] = 0;
1400                            fsck_send_msg(fsck_BMAPBADL2NLF, 2);
1401                    }
1402                    if (prms_ptr->buf_tree->dmt_leafidx != CTLLEAFIND) {
1403                            /* wrong 1st leaf index */
1404                            *other_errors = -1;
1405                            tree_spec_errors = -1;
1406                            msgprms[0] = message_parm_0;
1407                            msgprmidx[0] = prms_ptr->page_level;
1408                            sprintf(message_parm_1, "%d", prms_ptr->page_ordno);
1409                            msgprms[1] = message_parm_1;
1410                            msgprmidx[1] = 0;
1411                            fsck_send_msg(fsck_BMAPBADLFI, 2);
1412                    }
1413                    if (prms_ptr->buf_tree->dmt_height != 5) {
1414                            /* wrong stree height */
1415                            *other_errors = -1;
1416                            tree_spec_errors = -1;
1417                            msgprms[0] = message_parm_0;
1418                            msgprmidx[0] = prms_ptr->page_level;
1419                            sprintf(message_parm_1, "%d", prms_ptr->page_ordno);
1420                            msgprms[1] = message_parm_1;
1421                            msgprmidx[1] = 0;
1422                            fsck_send_msg(fsck_BMAPBADHT, 2);
1423                    }
1424                    if (prms_ptr->buf_tree->dmt_budmin != (L2BPERDMAP + level * L2LPERCTL)) {
1425                            /* wrong min buddy value */
1426                            *other_errors = -1;
1427                            tree_spec_errors = -1;
1428                            msgprms[0] = message_parm_0;
1429                            msgprmidx[0] = prms_ptr->page_level;
1430                            sprintf(message_parm_1, "%d", prms_ptr->page_ordno);
1431                            msgprms[1] = message_parm_1;
1432                            msgprmidx[1] = 0;
1433                            fsck_send_msg(fsck_BMAPBADBMN, 2);
1434                    }
1435
1436                    /*
1437                     * if we found errors in the fields which specify the summary
```

```
                 * tree then we won't take the time to verify the tree itself.
                 *
                 * (The errors already detected would corrupt the summary tree,
                 *  so info about the bad tree would only be noise at this point.)
                 */
                if (!tree_spec_errors) {
                        /* tree specification fields are ok */
                        prms_ptr->buf_stree =
                            &(prms_ptr->buf_tree->dmt_stree[0]);
                        prms_ptr->nleafs = LPERCTL;
                        prms_ptr->budmin = (L2BPERDMAP + level * L2LPERCTL);
                        prms_ptr->l2nleafs = L2LPERCTL;
                        prms_ptr->leafidx = CTLLEAFIND;

                        blsv_rc = stree_verify(prms_ptr, root_data);

                }
        }
        return (blsv_rc);
}

/*****************************************************************************
 * NAME: rebuild_blkall_map
 *
 * FUNCTION:  Rebuild the JFS Aggregate Block Map in the aggregate.
 *
 * PARAMETERS:  none
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int rebuild_blkall_map()
{
        int rbam_rc = FSCK_OK;
        int8_t sumtree_root_data;
        uint32_t leafidx;

#define MAXIDX (LPERCTL - 1)

        rbam_rc = init_bmap_info();

        /*
         * since the dmap I/O buffer is really the same storage as the
         * IAG I/O buffer, flush out any pending writes that may remain
         * from IAG processing.
         */
        rbam_rc = iags_flush();

        /*
         * rebuild the dmap pages.  Rebuild each L0 and L1 page
         * if and when the information for is complete.
         */
        while ((rbam_rc == FSCK_OK)
               && (bmap_recptr->dmappg_ordno < bmap_recptr->dmappg_count)) {

                rbam_rc = dmappg_rebuild(&sumtree_root_data);
                if (rbam_rc == FSCK_OK) {
                        /*
                         * the data in the dmap summary tree root goes into a leaf of
                         * the current L0 page
                         */
                        bmap_recptr->L0_wsp_sleafs[bmap_recptr->dmappg_idx] =
                            sumtree_root_data;

                        /* move to next dmap page */
                        bmap_recptr->dmappg_ordno++;
                        bmap_recptr->dmap_1stblk += BPERDMAP;

                        if (bmap_recptr->dmappg_idx < MAXIDX) {
                                /* still gathering info about this L0 page */
                                bmap_recptr->dmappg_idx++;
                        } else {
                                /* we have all the info needed for the current L0 page */
                                bmap_recptr->dmappg_idx = 0;
                                rbam_rc =
                                    Ln_tree_rebuild(0, bmap_recptr->L0pg_1stblk,
                                                    &(bmap_recptr->L0_bufptr),
                                                    &sumtree_root_data);

                                if (rbam_rc == FSCK_OK) {
                                        /*
                                         * the data in the L0 summary tree root goes into
                                         * a leaf of the current L1 page
                                         */
                                        bmap_recptr->L1_wsp_sleafs[bmap_recptr->
                                                                   L0pg_idx] =
                                            sumtree_root_data;
                                        /* move to the next L0 page */
                                        bmap_recptr->L0pg_ordno++;
```

```
1528                                    bmap_recptr->L0pg_1stblk =
1529                                        bmap_recptr->dmap_1stblk;
1530
1531                                    if (bmap_recptr->L0pg_idx < MAXIDX) {
1532                                            /* still gathering info about this L1 page */
1533                                            bmap_recptr->L0pg_idx++;
1534                                    } else {
1535                                            /* we have all the info needed for the current L1 page */
1536                                            bmap_recptr->L0pg_idx = 0;
1537                                            rbam_rc =
1538                                                Ln_tree_rebuild(1,
1539                                                                bmap_recptr->
1540                                                                L1pg_1stblk,
1541                                                                &
1542                                                                (bmap_recptr->
1543                                                                 L1_bufptr),
1544                                                                &sumtree_root_data);
1545
1546                                            if (rbam_rc == FSCK_OK) {
1547                                                    /*
1548                                                     * the data in the L1 summary tree root goes into
1549                                                     * a leaf of the current L2 page
1550                                                     */
1551                                                    bmap_recptr->
1552                                                        L2_wsp_sleafs
1553                                                        [bmap_recptr->
1554                                                         L1pg_idx] =
1555                                                        sumtree_root_data;
1556
1557                                                    /* move to the next L1 page */
1558                                                    bmap_recptr->L1pg_ordno++;
1559                                                    bmap_recptr->
1560                                                        L1pg_1stblk =
1561                                                        bmap_recptr->
1562                                                        dmap_1stblk;
1563
1564                                                    /*
1565                                                     * note that there is always AT MOST a single L2 page
1566                                                     */
1567                                                    bmap_recptr->L1pg_idx++;
1568
1569                                            }
1570                                    }
1571                            }
1572                    }
1573            }
1574    }
1575
1576    /*
1577     * finish up the partial pages
1578     */
1579
1580    if (rbam_rc == FSCK_OK) {
1581            if (bmap_recptr->dmappg_idx != 0) {
1582                    /* there's a partial L0 page */
1583                    for (leafidx = bmap_recptr->dmappg_idx;
1584                         (leafidx <= MAXIDX); leafidx++) {
1585                            bmap_recptr->L0_wsp_sleafs[leafidx] = -1;
1586                    }
1587                    rbam_rc = Ln_tree_rebuild(0, bmap_recptr->L0pg_1stblk,
1588                                              &(bmap_recptr->L0_bufptr),
1589                                              &sumtree_root_data);
1590                    if (rbam_rc == FSCK_OK) {
1591                            /*
1592                             * the data in the L0 summary tree root goes into
1593                             * a leaf of the current L1 page
1594                             */
1595                            bmap_recptr->L1_wsp_sleafs[bmap_recptr->
1596                                                       L0pg_idx] =
1597                                sumtree_root_data;
1598                            bmap_recptr->L0pg_idx++;
1599                    }
1600            }
1601    }
1602    if (rbam_rc == FSCK_OK) {
1603            if ((bmap_recptr->L1pg_count > 0) && (bmap_recptr->L0pg_idx != 0)) {
1604                    /* there's enough data for an L1 level, and there's a partial L1 page */
1605                    for (leafidx = bmap_recptr->L0pg_idx;
1606                         (leafidx <= MAXIDX); leafidx++) {
1607                            bmap_recptr->L1_wsp_sleafs[leafidx] = -1;
1608                    }
1609                    rbam_rc = Ln_tree_rebuild(1, bmap_recptr->L1pg_1stblk,
1610                                              &(bmap_recptr->L1_bufptr),
1611                                              &sumtree_root_data);
1612                    if (rbam_rc == FSCK_OK) {
1613                            /*
1614                             * the data in the L0 summary tree root goes into
1615                             * a leaf of the current L1 page
1616                             */
1617                            bmap_recptr->L2_wsp_sleafs[bmap_recptr->
```

```
                                                        L1pg_idx] =
                                    sumtree_root_data;
                                bmap_recptr->L1pg_idx++;
                        }
                }
        }
        if (rbam_rc == FSCK_OK) {
                if ((bmap_recptr->L2pg_count > 0) && (bmap_recptr->L1pg_idx != 0)) {
                        /* there's enough data for an L2 level, and there's a partial L2 page */
                        for (leafidx = bmap_recptr->L1pg_idx;
                             (leafidx <= MAXIDX); leafidx++) {
                                bmap_recptr->L2_wsp_sleafs[leafidx] = -1;
                        }
                        rbam_rc = Ln_tree_rebuild(2, bmap_recptr->L2pg_1stblk,
                                                  &(bmap_recptr->L2_bufptr),
                                                  &sumtree_root_data);
                }
        }
        /*
         * Now go verify the Block Allocation Map Control page
         */
        if (rbam_rc == FSCK_OK) {
                rbam_rc = ctlpage_rebuild(sumtree_root_data);
        }
        return (rbam_rc);
}

/*****************************************************************************
 * NAME: stree_rebuild
 *
 * FUNCTION:  Rebuild the specified summary tree.
 *
 * PARAMETERS:
 *      prms_ptr   - input - pointer to a data area describing the tree
 *                          to rebuild and containing the dmap which the
 *                          tree summarizes.
 *      root_data  - input - pointer to a variable in which to return the
 *                          data value stored in the root node of the tree
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int stree_rebuild(struct fsck_stree_proc_parms *prms_ptr, int8_t * root_data)
{
        int bsr_rc = FSCK_OK;
        uint32_t node_idx, last_leaf_idx;

        /*
         * copy the leaf data into the buffer
         */
        last_leaf_idx = prms_ptr->leafidx + prms_ptr->nleafs - 1;
        for (node_idx = prms_ptr->leafidx; (node_idx <= last_leaf_idx);
             node_idx++) {
                prms_ptr->buf_stree[node_idx] = prms_ptr->wsp_stree[node_idx];
        }

        /*
         * build the summary tree from the "raw" leaf values
         */
        *root_data =
            ujfs_adjtree(prms_ptr->buf_stree, prms_ptr->l2nleafs,
                         prms_ptr->budmin);

        return (bsr_rc);
}

/*****************************************************************************
 * NAME: stree_verify
 *
 * FUNCTION:  Verify the specified summary tree.
 *
 * PARAMETERS:
 *      prms_ptr   - input - pointer to a data area describing the tree
 *                          to verify and containing the dmap which the
 *                          tree summarizes.
 *      root_data  - input - pointer to a variable in which to return the
 *                          data value which should be in the root node
 *                          of the tree (and may in fact be in the root
 *                          node of the tree)
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int stree_verify(struct fsck_stree_proc_parms *prms_ptr, int8_t * root_data)
{
        int bsv_rc = FSCK_OK;
        uint32_t node_idx, last_leaf_idx;

```

```
1708          /*
1709           * build the summary tree from the "raw" leaf values
1710           */
1711          *root_data =
1712              ujfs_adjtree(prms_ptr->wsp_stree, prms_ptr->l2nleafs,
1713                           prms_ptr->budmin);
1714
1715          /*
1716           * Now see if the tree in the buffer matches the one we just
1717           * built in the workspace.
1718           *
1719           * We distinguish between incorrect internal nodes and incorrect
1720           * leaf nodes because they can be symptoms of different problems.
1721           */
1722          for (node_idx = 0; (node_idx < prms_ptr->leafidx); node_idx++) {
1723                  if (prms_ptr->buf_stree[node_idx] != prms_ptr->wsp_stree[node_idx]) {
1724                          /* they don't match! */
1725                          *(prms_ptr->intval_error) = -1;
1726                          sprintf(message_parm_0, "%d", node_idx);
1727                          msgprms[0] = message_parm_0;
1728                          msgprmidx[0] = 0;
1729                          msgprms[1] = message_parm_1;
1730                          msgprmidx[1] = prms_ptr->page_level;
1731                          sprintf(message_parm_2, "%d", prms_ptr->page_ordno);
1732                          msgprms[2] = message_parm_2;
1733                          msgprmidx[2] = 0;
1734                          fsck_send_msg(fsck_BMAPBADLNV, 3);
1735                  }
1736          }
1737
1738          last_leaf_idx = prms_ptr->leafidx + prms_ptr->nleafs - 1;
1739          for (node_idx = prms_ptr->leafidx; (node_idx <= last_leaf_idx);
1740               node_idx++) {
1741                  if (prms_ptr->buf_stree[node_idx] != prms_ptr->wsp_stree[node_idx]) {
1742                          /* they don't match! */
1743                          *(prms_ptr->lfval_error) = -1;
1744                          sprintf(message_parm_0, "%d", node_idx);
1745                          msgprms[0] = message_parm_0;
1746                          msgprmidx[0] = 0;
1747                          msgprms[1] = message_parm_1;
1748                          msgprmidx[1] = prms_ptr->page_level;
1749                          sprintf(message_parm_2, "%d", prms_ptr->page_ordno);
1750                          msgprms[2] = message_parm_2;
1751                          msgprmidx[2] = 0;
1752                          fsck_send_msg(fsck_BMAPBADLFV, 3);
1753                  }
1754          }
1755
1756          return (bsv_rc);
1757  }
1758
1759  /*****************************************************************************
1760   * NAME: verify_blkall_map
1761   *
1762   * FUNCTION:  Validate the JFS Aggregate Block Map for the aggregate.
1763   *
1764   * PARAMETERS:  none
1765   *
1766   * RETURNS:
1767   *      success: FSCK_OK
1768   *      failure: something else
1769   */
1770  int verify_blkall_map()
1771  {
1772          int vbam_rc = FSCK_OK;
1773          int8_t sumtree_root_data;
1774          uint32_t leafidx;
1775
1776  #define MAXIDX (LPERCTL - 1)
1777
1778          vbam_rc = init_bmap_info();
1779
1780          /*
1781           * since the dmap I/O buffer is really the same storage as the
1782           * IAG I/O buffer, flush out any pending writes that may remain
1783           * from IAG processing.
1784           */
1785          vbam_rc = iags_flush();
1786
1787          /*
1788           * Verify the dmap pages.  Verify each L0 and L1 page
1789           * if and when the information for is complete.
1790           */
1791          while ((vbam_rc == FSCK_OK)
1792                 && (bmap_recptr->dmappg_ordno < bmap_recptr->dmappg_count)) {
1793
1794                  vbam_rc = dmappg_verify(&sumtree_root_data);
1795                  if (vbam_rc == FSCK_OK) {
1796                          /*
1797                           * the data in the dmap summary tree root goes into a leaf of
```

```
1798                            * the current L0 page
1799                            */
1800                           bmap_recptr->L0_wsp_sleafs[bmap_recptr->dmappg_idx] =
1801                               sumtree_root_data;
1802                           /* move to next dmap page */
1803                           bmap_recptr->dmappg_ordno++;
1804                           bmap_recptr->dmap_1stblk += BPERDMAP;
1805
1806                           if (bmap_recptr->dmappg_idx < MAXIDX) {
1807                                   /* still gathering info about this L0 page */
1808                                   bmap_recptr->dmappg_idx++;
1809                           } else {
1810                                   /* we have all the info needed for the current L0 page */
1811                                   bmap_recptr->dmappg_idx = 0;
1812                                   vbam_rc =
1813                                       Ln_tree_verify(0, bmap_recptr->L0pg_1stblk,
1814                                                      &(bmap_recptr->L0_bufptr),
1815                                                      &sumtree_root_data);
1816
1817                                   if (vbam_rc == FSCK_OK) {
1818                                           /*
1819                                            * the data in the L0 summary tree root goes into
1820                                            * a leaf of the current L1 page
1821                                            */
1822                                           bmap_recptr->L1_wsp_sleafs[bmap_recptr->
1823                                                                      L0pg_idx] =
1824                                               sumtree_root_data;
1825                                           /* move to the next L0 page */
1826                                           bmap_recptr->L0pg_ordno++;
1827                                           bmap_recptr->L0pg_1stblk =
1828                                               bmap_recptr->dmap_1stblk;
1829                                           if (bmap_recptr->L0pg_idx < MAXIDX) {
1830                                                   /* still gathering info about this L1 page */
1831                                                   bmap_recptr->L0pg_idx++;
1832                                           } else {
1833                                                   /* we have all the info needed for the current L1 page */
1834                                                   bmap_recptr->L0pg_idx = 0;
1835                                                   vbam_rc =
1836                                                       Ln_tree_verify(1,
1837                                                                      bmap_recptr->
1838                                                                      L1pg_1stblk,
1839                                                                      &
1840                                                                      (bmap_recptr->
1841                                                                       L1_bufptr),
1842                                                                      &sumtree_root_data);
1843                                                   if (vbam_rc == FSCK_OK) {
1844                                                           /*
1845                                                            * the data in the L1 summary tree root goes into
1846                                                            * a leaf of the current L2 page
1847                                                            */
1848                                                           bmap_recptr->
1849                                                               L2_wsp_sleafs
1850                                                               [bmap_recptr->
1851                                                                L1pg_idx] =
1852                                                               sumtree_root_data;
1853                                                           /* move to the next L1 page */
1854                                                           bmap_recptr->L1pg_ordno++;
1855                                                           bmap_recptr->
1856                                                               L1pg_1stblk =
1857                                                               bmap_recptr->
1858                                                               dmap_1stblk;
1859                                                           /*
1860                                                            * note that there is always AT MOST a single L2 page
1861                                                            */
1862                                                           bmap_recptr->L1pg_idx++;
1863                                                   }
1864                                           }
1865                                   }
1866                           }
1867                   }
1868           }
1869           /*
1870            * finish up the partial pages
1871            */
1872           if (vbam_rc == FSCK_OK) {
1873                   if (bmap_recptr->dmappg_idx != 0) {
1874                           /* there's a partial L0 page */
1875                           for (leafidx = bmap_recptr->dmappg_idx;
1876                                (leafidx <= MAXIDX); leafidx++) {
1877                                   bmap_recptr->L0_wsp_sleafs[leafidx] = -1;
1878                           }
1879                           vbam_rc = Ln_tree_verify(0, bmap_recptr->L0pg_1stblk,
1880                                                    &(bmap_recptr->L0_bufptr),
1881                                                    &sumtree_root_data);
1882                           if (vbam_rc == FSCK_OK) {
1883                                   /*
1884                                    * the data in the L0 summary tree root goes into
1885                                    * a leaf of the current L1 page
1886                                    */
1887                                   bmap_recptr->L1_wsp_sleafs[bmap_recptr->
```

```
                                                        L0pg_idx] =
                                    sumtree_root_data;
                                bmap_recptr->L0pg_idx++;
                        }
                }
        }
        if (vbam_rc == FSCK_OK) {
                if ((bmap_recptr->L1pg_count > 0) && (bmap_recptr->L0pg_idx != 0)) {
                        /* there's enough data for an L1 level, and there's a partial L1 page */
                        for (leafidx = bmap_recptr->L0pg_idx;
                             (leafidx <= MAXIDX); leafidx++) {
                                bmap_recptr->L1_wsp_sleafs[leafidx] = -1;
                        }       /* end for leafidx */

                        vbam_rc = Ln_tree_verify(1, bmap_recptr->L1pg_1stblk,
                                                 &(bmap_recptr->L1_bufptr),
                                                 &sumtree_root_data);
                        if (vbam_rc == FSCK_OK) {
                                /*
                                 * the data in the L0 summary tree root goes into
                                 * a leaf of the current L1 page
                                 */
                                bmap_recptr->L2_wsp_sleafs[bmap_recptr->
                                                        L1pg_idx] =
                                    sumtree_root_data;
                                bmap_recptr->L1pg_idx++;
                        }
                }
        }
        if (vbam_rc == FSCK_OK) {
                if ((bmap_recptr->L2pg_count > 0) && (bmap_recptr->L1pg_idx != 0)) {
                        /* there's enough data for an L2 level, and there's a partial L2 page */
                        for (leafidx = bmap_recptr->L1pg_idx;
                             (leafidx <= MAXIDX); leafidx++) {
                                bmap_recptr->L2_wsp_sleafs[leafidx] = -1;
                        }

                        vbam_rc = Ln_tree_verify(2, bmap_recptr->L2pg_1stblk,
                                                 &(bmap_recptr->L2_bufptr),
                                                 &sumtree_root_data);
                }
        }
        /*
         * Now go verify the Block Allocation Map Control page
         */
        if (vbam_rc == FSCK_OK) {
                vbam_rc = ctlpage_verify(sumtree_root_data);
        }
        /*
         * issue summary messages about the Block Allocation Map validation
         */
        if (vbam_rc == FSCK_OK) {
                vbam_rc = verify_blkall_summary_msgs();
        }
        return (vbam_rc);
}

/*****************************************************************************
 * NAME: verify_blkall_summary_msgs
 *
 * FUNCTION: Issue summary messages with the results of JFS Aggregate Block
 *           Map validation.
 *
 * PARAMETERS:  none
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int verify_blkall_summary_msgs()
{
        int vbsm_rc = FSCK_OK;

        if (bmap_recptr->dmap_pmap_error) {
                fsck_send_msg(fsck_BADDMAPPMAPS, 0);
        }
        if (bmap_recptr->dmap_slfv_error) {
                msgprms[0] = message_parm_0;
                msgprmidx[0] = fsck_dmap;
                fsck_send_msg(fsck_BADBMAPSLFV, 1);
        }
        if (bmap_recptr->dmap_slnv_error) {
                msgprms[0] = message_parm_0;
                msgprmidx[0] = fsck_dmap;
                fsck_send_msg(fsck_BADBMAPSLNV, 1);
        }
        if (bmap_recptr->dmap_other_error) {
                msgprms[0] = message_parm_0;
                msgprmidx[0] = fsck_dmap;
                fsck_send_msg(fsck_BADBMAPSOTHER, 1);
```

```
1978            }
1979            if (bmap_recptr->L0pg_slfv_error) {
1980                    msgprms[0] = message_parm_0;
1981                    msgprmidx[0] = fsck_L0;
1982                    fsck_send_msg(fsck_BADBMAPSLFV, 1);
1983            }
1984            if (bmap_recptr->L0pg_slnv_error) {
1985                    msgprms[0] = message_parm_0;
1986                    msgprmidx[0] = fsck_L0;
1987                    fsck_send_msg(fsck_BADBMAPSLNV, 1);
1988            }
1989            if (bmap_recptr->L0pg_other_error) {
1990                    msgprms[0] = message_parm_0;
1991                    msgprmidx[0] = fsck_L0;
1992                    fsck_send_msg(fsck_BADBMAPSOTHER, 1);
1993            }
1994            if (bmap_recptr->L1pg_slfv_error) {
1995                    msgprms[0] = message_parm_0;
1996                    msgprmidx[0] = fsck_L1;
1997                    fsck_send_msg(fsck_BADBMAPSLFV, 1);
1998            }
1999            if (bmap_recptr->L1pg_slnv_error) {
2000                    msgprms[0] = message_parm_0;
2001                    msgprmidx[0] = fsck_L1;
2002                    fsck_send_msg(fsck_BADBMAPSLNV, 1);
2003            }
2004            if (bmap_recptr->L1pg_other_error) {
2005                    msgprms[0] = message_parm_0;
2006                    msgprmidx[0] = fsck_L1;
2007                    fsck_send_msg(fsck_BADBMAPSOTHER, 1);
2008            }
2009            if (bmap_recptr->L2pg_slfv_error) {
2010                    msgprms[0] = message_parm_0;
2011                    msgprmidx[0] = fsck_L2;
2012                    fsck_send_msg(fsck_BADBMAPSLFV, 1);
2013            }
2014            if (bmap_recptr->L2pg_slnv_error) {
2015                    msgprms[0] = message_parm_0;
2016                    msgprmidx[0] = fsck_L2;
2017                    fsck_send_msg(fsck_BADBMAPSLNV, 1);
2018            }
2019            if (bmap_recptr->L2pg_other_error) {
2020                    msgprms[0] = message_parm_0;
2021                    msgprmidx[0] = fsck_L2;
2022                    fsck_send_msg(fsck_BADBMAPSOTHER, 1);
2023            }
2024            if (bmap_recptr->ctl_fctl_error) {
2025                    fsck_send_msg(fsck_BADBMAPCAGFCL, 0);
2026            }
2027            if (bmap_recptr->ctl_other_error) {
2028                    fsck_send_msg(fsck_BADBMAPCOTH, 0);
2029            }
2030            if (bmap_recptr->dmap_pmap_error || bmap_recptr->dmap_slfv_error ||
2031                bmap_recptr->dmap_slnv_error || bmap_recptr->dmap_other_error ||
2032                bmap_recptr->L0pg_slfv_error || bmap_recptr->L0pg_slnv_error ||
2033                bmap_recptr->L0pg_other_error || bmap_recptr->L1pg_slfv_error ||
2034                bmap_recptr->L1pg_slnv_error || bmap_recptr->L1pg_other_error ||
2035                bmap_recptr->L2pg_slfv_error || bmap_recptr->L2pg_slnv_error ||
2036                bmap_recptr->L2pg_other_error) {
2037                    agg_recptr->ag_dirty = 1;
2038                    fsck_send_msg(fsck_BADBLKALLOC, 0);
2039            }
2040            if (bmap_recptr->ctl_fctl_error || bmap_recptr->ctl_other_error) {
2041                    agg_recptr->ag_dirty = 1;
2042                    fsck_send_msg(fsck_BADBLKALLOCCTL, 0);
2043            }
2044            return (vbsm_rc);
2045    }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#include "xfsckint.h"

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * superblock buffer pointer
  *
  *      defined in xchkdsk.c
  */
extern struct superblock *sb_ptr;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * fsck aggregate info structure pointer
  *
  *      defined in xchkdsk.c
  */
extern struct fsck_agg_record *agg_recptr;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * For message processing
  *
  *      defined in xchkdsk.c
  */
extern char message_parm_0[];
extern char message_parm_1[];
extern char message_parm_2[];
extern char message_parm_3[];
extern char message_parm_4[];
extern char message_parm_5[];
extern char message_parm_6[];
extern char message_parm_7[];
extern char message_parm_8[];
extern char message_parm_9[];

extern char *msgprms[];
extern int16_t msgprmidx[];

extern char *verbose_msg_ptr;

extern char *MsgText[];

extern char *Vol_Label;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * For directory entry processing
  *
  *      defined in xchkdsk.c
  */
extern uint32_t key_len[2];
extern UniChar key[2][JFS_NAME_MAX];
extern UniChar ukey[2][JFS_NAME_MAX];

extern int32_t Uni_Name_len;
extern UniChar Uni_Name[JFS_NAME_MAX];

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * Device information.
  *
  *      defined in xchkdsk.c
  */
extern HFILE Dev_IOPort;
extern uint32_t Dev_blksize;

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * The current processing phase.
  *
  *      defined in xchkdsk.c
  *      constants defining possible values are defined in xfsck.h
  */
```

```
91  extern int current_fsck_phase;
92
93  /* VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
94   *
95   * The following are internal to this file
96   *
97   */
98  int adjust_parent(uint32_t, struct fsck_inode_record *, uint32_t);
99
100 int reset_parents(struct fsck_inode_record *, uint32_t);
101
102 /* VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV */
103
104 /*****************************************************************************
105  * NAME: adjust_parent
106  *
107  * FUNCTION: Add an fsck inode extension record to the parent inode's
108  *           fsck inode record so that the directory entry for the
109  *           given child will be removed.
110  *
111  * PARAMETERS:
112  *      child_ino        − input − ordinal number of child inode
113  *      child_inorecptr  − input − pointer to an fsck record describing the
114  *                                 child inode
115  *      parent_ino       − input − ordinal number of parent inode
116  *
117  * RETURNS:
118  *      success: FSCK_OK
119  *      failure: something else
120  */
121 int adjust_parent(uint32_t child_ino, struct fsck_inode_record *child_inorecptr,
122                   uint32_t parent_ino)
123 {
124         int adjp_rc = FSCK_OK;
125         struct fsck_inode_ext_record *this_ext;
126         struct fsck_inode_record *parent_inorecptr;
127         int is_aggregate = 0;
128         int alloc_ifnull = 0;
129
130         adjp_rc =
131             get_inorecptr(is_aggregate, alloc_ifnull, parent_ino,
132                           &parent_inorecptr);
133
134         if ((adjp_rc == FSCK_OK) && (parent_inorecptr == NULL)) {
135                 adjp_rc = FSCK_INTERNAL_ERROR_4;
136                 sprintf(message_parm_0, "%d", adjp_rc);
137                 msgprms[0] = message_parm_0;
138                 msgprmidx[0] = 0;
139                 sprintf(message_parm_1, "%d", child_ino);
140                 msgprms[1] = message_parm_1;
141                 msgprmidx[1] = 0;
142                 sprintf(message_parm_2, "%d", parent_ino);
143                 msgprms[2] = message_parm_2;
144                 msgprmidx[2] = 0;
145                 sprintf(message_parm_3, "%d", 0);
146                 msgprms[3] = message_parm_3;
147                 msgprmidx[3] = 0;
148                 fsck_send_msg(fsck_INTERNALERROR, 4);
149
150         } else if (adjp_rc == FSCK_OK) {
151                 /* located the parent's inode record */
152                 adjp_rc = get_inode_extension(&this_ext);
153                 if (adjp_rc == FSCK_OK) {
154                         /* got extension */
155                         this_ext->ext_type = rmv_direntry_extension;
156                         this_ext->inonum = child_ino;
157                         this_ext->next = parent_inorecptr->ext_rec;
158                         this_ext->ino_type = child_inorecptr->inode_type;
159                         parent_inorecptr->ext_rec = this_ext;
160                         parent_inorecptr->adj_entries = 1;
161                         agg_recptr->corrections_needed = 1;
162                 }
163         }
164         return (adjp_rc);
165 }
166
167 /*****************************************************************************
168  * NAME: adjust_parents
169  *
170  * FUNCTION: Add an fsck inode extension record, to the fsck inode record
171  *           of each of the given inode's parent inodes, so that all
172  *           directory entries for the given inode will be removed.  The
173  *           exception is that, if the given inode is a directory inode,
174  *           the expected parent may be omitted from this processing.
175  *
176  * PARAMETERS:
177  *      ino_recptr  − input − pointer to an fsck record describing the inode
178  *      ino_idx     − input − ordinal number of the inode
179  *
180  * RETURNS:
```

```
 *      success: FSCK_OK
 *      failure: something else
 */
int adjust_parents(struct fsck_inode_record *ino_recptr, uint32_t ino_idx)
{
        int adjps_rc = FSCK_OK;
        struct fsck_inode_ext_record *this_ext;
        struct fsck_inode_ext_record *rest_of_list;
        int keep_primary_parent = 0;

        /*
         * if this is a directory with illegal hard links then the
         * inode number in the fsck inode record is the one stored in
         * the inode on disk.  Then if the inode isn't being released
         * and any observed parent matches the stored parent, that
         * parent will not be adjusted.
         */
        if ((ino_recptr->inode_type == directory_inode)
            && (ino_recptr->unxpctd_prnts)) {
                /* dir with multiple parents */
                if (!ino_recptr->selected_to_rls) {
                        /* not being released */
                        if ((ino_recptr->parent_inonum != ROOT_I)
                            || (!agg_recptr->rootdir_rebuilt)) {
                                keep_primary_parent = 1;
                        }
                }
        } else {
                /* not a dir with multiple parents -- this must be either an
                 * inode approved for released or an unallocated inode with
                 * parents observed.
                 */
                /*
                 * the 1st parent observed is in the inode record.  Any others are
                 * in extension records.
                 */
                if (ino_recptr->parent_inonum != 0) {
                        if ((ino_recptr->parent_inonum != ROOT_I)
                            || (!agg_recptr->rootdir_rebuilt)) {
                                /* either this parent isn't the root or
                                   else the root dir has not been rebuilt */
                                adjps_rc =
                                    adjust_parent(ino_idx, ino_recptr,
                                                  ino_recptr->parent_inonum);
                                ino_recptr->parent_inonum = 0;  /* clear it */
                        }
                }
        }

        /*
         * detach the extensions list from the inode record
         */
        this_ext = ino_recptr->ext_rec;
        ino_recptr->ext_rec = NULL;

        while ((adjps_rc == FSCK_OK) && (this_ext != NULL)) {
                /* there may be more parents */
                rest_of_list = this_ext->next;
                if (this_ext->ext_type != parent_extension) {
                        /* not a parent */
                        this_ext->next = ino_recptr->ext_rec;
                        ino_recptr->ext_rec = this_ext;
                } else {
                        /* parent extension */
                        if ((this_ext->inonum == ROOT_I)
                            && (agg_recptr->rootdir_rebuilt)) {
                                /*
                                 * This parent is the root and the root dir has
                                 * been rebuilt.  This is equivalent to a parent
                                 * marked for release.
                                 */
                                release_inode_extension(this_ext);

                        } else if ((keep_primary_parent) &&
                                   (this_ext->inonum ==
                                    ino_recptr->parent_inonum)) {
                                /*
                                 * We're keeping the entry for the expected parent.
                                 * Just drop the extension record and clear the
                                 * 'unexpected parents' flag.  When we finish this
                                 * routine the inode will be all set.
                                 */
                                release_inode_extension(this_ext);
                                ino_recptr->unxpctd_prnts = 0;

                        } else {
                                /* either not keeping the 'primary'
                                   or else this isn't it */
                                adjps_rc =
                                    adjust_parent(ino_idx, ino_recptr,
```

```
                                                this_ext->inonum);
                                release_inode_extension(this_ext);
                        }
                }

                this_ext = rest_of_list;
        }
        return (adjps_rc);
}

/*****************************************************************************
 * NAME: check_connectedness
 *
 * FUNCTION:  Verify that, after approved corrections are made, all inodes
 *            in use will be connected to the root directory tree.
 *
 * PARAMETERS:  none
 *
 * NOTES:  o A directory inode must have exactly one parent inode.
 *         o A non-directory inode must have at least one parent inode.
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int check_connectedness()
{
        int cc_rc = FSCK_OK;
        uint32_t ino_idx;
        struct fsck_inode_record *this_inorec;
        struct fsck_inode_ext_record *new_ext;
        int aggregate_inode = 0;

        struct fsck_ino_msg_info ino_msg_info;
        struct fsck_ino_msg_info *msg_info_ptr;

        msg_info_ptr = &ino_msg_info;
        /* all fileset owned */
        msg_info_ptr->msg_inopfx = fsck_fset_inode;

        /*
         * detect orphan inodes, including ones which we're about to
         * orphan by releasing inodes.
         *
         * if any non-orphan inode was flagged as a directory with illegal
         * hard links, see if it's going to be true after we release inodes.
         * If not, make sure the remaining link matches the one stored in
         * the inode.
         */
        cc_rc = get_inorecptr_first(aggregate_inode, &ino_idx, &this_inorec);

        while ((cc_rc == FSCK_OK) && (this_inorec != NULL)
               && (ino_idx < FILESET_OBJECT_I)) {
                /*
                 * not interesting until we get past the root inode
                 * and the special fileset inodes.
                 */
                cc_rc =
                    get_inorecptr_next(aggregate_inode, &ino_idx, &this_inorec);
        }

        while ((cc_rc == FSCK_OK) && (this_inorec != NULL)) {

                if ((this_inorec->in_use) && (!this_inorec->selected_to_rls)) {
                        /* inode in use and not selected to release */
                        msg_info_ptr->msg_inonum = ino_idx;
                        if (this_inorec->inode_type == directory_inode) {
                                msg_info_ptr->msg_inotyp = fsck_directory;

                        } else if (this_inorec->inode_type == symlink_inode) {
                                msg_info_ptr->msg_inotyp = fsck_symbolic_link;

                        } else if (this_inorec->inode_type ==
                                   char_special_inode) {
                                msg_info_ptr->msg_inotyp = fsck_char_special;

                        } else if (this_inorec->inode_type ==
                                   block_special_inode) {
                                msg_info_ptr->msg_inotyp = fsck_block_special;

                        } else if (this_inorec->inode_type == FIFO_inode) {
                                msg_info_ptr->msg_inotyp = fsck_FIFO;

                        } else if (this_inorec->inode_type == SOCK_inode) {
                                msg_info_ptr->msg_inotyp = fsck_SOCK;

                        } else {
                                /* a regular file */
                                msg_info_ptr->msg_inotyp = fsck_file;
                        }
```

```
361
362                     if (this_inorec->parent_inonum == 0) {
363                             /* no parents were observed by fsck */
364                             msgprms[0] = message_parm_0;
365                             msgprmidx[0] = msg_info_ptr->msg_inopfx;
366                             sprintf(message_parm_1, "%d",
367                                     msg_info_ptr->msg_inonum);
368                             msgprms[1] = message_parm_1;
369                             msgprmidx[1] = 0;
370                             fsck_send_msg(fsck_INONOPATHS, 2);
371                     } else {
372                             /* at least one parent observed by fsck */
373                             /*
374                              * make adjustments to child records for
375                              * parents which will be released
376                              */
377                             cc_rc = reset_parents(this_inorec, ino_idx);
378                     }
379
380                     if (cc_rc == FSCK_OK) {
381                             if (this_inorec->parent_inonum == 0) {
382                                     /* unconnected !! */
383                                     /*
384                                      * do not issue a message to inform the user about
385                                      * this condition since it is a side effect of
386                                      * the (approved) release of some other inode(s)
387                                      */
388                                     this_inorec->reconnect = 1;
389                                     agg_recptr->corrections_approved = 1;
390                                     cc_rc = get_inode_extension(&new_ext);
391                                     if (cc_rc == FSCK_OK) {
392                                             /* got an extension record */
393                                             new_ext->ext_type =
394                                                 add_direntry_extension;
395                                             new_ext->inonum = ino_idx;
396                                             new_ext->ino_type =
397                                                 this_inorec->inode_type;
398                                             new_ext->next =
399                                                 agg_recptr->
400                                                 inode_reconn_extens;
401                                             agg_recptr->
402                                                 inode_reconn_extens =
403                                                 new_ext;
404                                             /* increment for the link from
405                                              * parent after reconnect
406                                              */
407                                             this_inorec->link_count++;
408                                     }
409                             } else {
410                                     /* else still connected */
411                                     if (this_inorec->unxpctd_prnts) {
412                                             /* multiple parents */
413                                             cc_rc =
414                                                 display_paths(ino_idx,
415                                                               this_inorec,
416                                                               msg_info_ptr);
417
418                                             if (cc_rc == FSCK_OK) {
419
420                                                     if (agg_recptr->
421                                                         processing_readwrite)
422                                                     {
423                                                             cc_rc =
424                                                                 adjust_parents
425                                                                 (this_inorec,
426                                                                  ino_idx);
427                                                             msgprms[0] =
428                                                                 message_parm_0;
429                                                             msgprmidx[0] =
430                                                                 msg_info_ptr->
431                                                                 msg_inopfx;
432                                                             sprintf
433                                                                 (message_parm_1,
434                                                                  "%d",
435                                                                  msg_info_ptr->
436                                                                  msg_inonum);
437                                                             msgprms[1] =
438                                                                 message_parm_1;
439                                                             msgprmidx[1] =
440                                                                 0;
441                                                             fsck_send_msg
442                                                                 (fsck_WILLFIXDIRWHDLKS,
443                                                                  2);
444
445                                                     } else {
446                                                             /* no write access */
447                                                             this_inorec->
448                                                                 unxpctd_prnts
449                                                                 = 0;
450                                                             agg_recptr->
```

```
                                                        ag_dirty =
                                                        1;
                                                    msgprms[0] =
                                                        message_parm_0;
                                                    msgprmidx[0] =
                                                        msg_info_ptr->
                                                        msg_inopfx;
                                                    sprintf
                                                        (message_parm_1,
                                                         "%d",
                                                         msg_info_ptr->
                                                         msg_inonum);
                                                    msgprms[1] =
                                                        message_parm_1;
                                                    msgprmidx[1] =
                                                        0;
                                                    fsck_send_msg
                                                        (fsck_DIRWHDLKS,
                                                         2);
                                                }
                                            }
                                        } else if ((cc_rc == FSCK_OK)
                                                   && (this_inorec->
                                                       crrct_prnt_inonum)) {
                                            /*
                                             * a single parent but not the one
                                             * named in the implied '..' entry
                                             */
                                            cc_rc =
                                                display_paths(ino_idx,
                                                              this_inorec,
                                                              msg_info_ptr);

                                            if (agg_recptr->
                                                processing_readwrite) {
                                                msgprms[0] =
                                                    message_parm_0;
                                                msgprmidx[0] =
                                                    msg_info_ptr->
                                                    msg_inopfx;
                                                sprintf(message_parm_1,
                                                        "%d",
                                                        msg_info_ptr->
                                                        msg_inonum);
                                                msgprms[1] =
                                                    message_parm_1;
                                                msgprmidx[1] = 0;
                                                fsck_send_msg
                                                    (fsck_WILLFIXINCREF,
                                                     2);
                                            } else {
                                                /* no write access */
                                                this_inorec->
                                                    crrct_prnt_inonum =
                                                    0;
                                                agg_recptr->ag_dirty =
                                                    1;
                                                msgprms[0] =
                                                    message_parm_0;
                                                msgprmidx[0] =
                                                    msg_info_ptr->
                                                    msg_inopfx;
                                                sprintf(message_parm_1,
                                                        "%d",
                                                        msg_info_ptr->
                                                        msg_inonum);
                                                msgprms[1] =
                                                    message_parm_1;
                                                msgprmidx[1] = 0;
                                                fsck_send_msg
                                                    (fsck_INCINOREF, 2);
                                            }
                                        }
                                    }
                                }
                            }
                            if (cc_rc == FSCK_OK) {
                                cc_rc =
                                    get_inorecptr_next(aggregate_inode, &ino_idx,
                                                       &this_inorec);
                            }
                        }
                        return (cc_rc);
                    }

                    /*****************************************************************************
                     * NAME: check_dir_integrity
                     *
                     * FUNCTION:  Verify that no directory has more than 1 entry for any
                     *            single inode.  If a directory does, then that directory
```

```
 *            is corrupt.
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int check_dir_integrity()
{
        int cdi_rc = FSCK_OK;
        uint32_t ino_idx;
        struct fsck_inode_record *this_inorec;
        struct fsck_inode_record *that_inorec;
        int aggregate_inode = 0;
        int alloc_ifnull = 0;
        struct fsck_inode_ext_record *this_ext;
        struct fsck_inode_ext_record *that_ext;
        int dup_parent_detected = 0;

        struct fsck_ino_msg_info ino_msg_info;
        struct fsck_ino_msg_info *msg_info_ptr;

        msg_info_ptr = &ino_msg_info;
        /* all fileset owned */
        msg_info_ptr->msg_inopfx = fsck_fset_inode;

        /*
         * Verify that no inode has multiple links from the same
         * directory.
         */
        cdi_rc = get_inorecptr_first(aggregate_inode, &ino_idx, &this_inorec);

        while ((cdi_rc == FSCK_OK) && (this_inorec != NULL)
               && (ino_idx < FILESET_OBJECT_I)) {
                /*
                 * not interesting until we get past the root inode
                 * and the special fileset inodes.
                 */
                cdi_rc =
                    get_inorecptr_next(aggregate_inode, &ino_idx, &this_inorec);
        }

        while ((cdi_rc == FSCK_OK) && (this_inorec != NULL)) {
                if ((this_inorec->in_use) && (!this_inorec->selected_to_rls)
                    && (this_inorec->inode_type == directory_inode)) {
                        /* directory inode in use and not selected to release */
                        msg_info_ptr->msg_inotyp = fsck_directory;
                        msg_info_ptr->msg_inonum = ino_idx;

                        if (this_inorec->parent_inonum != 0) {
                                /* at least 1 parent observed by fsck */
                                if (this_inorec->ext_rec) {
                                        /* maybe more parents have been observed */
                                        /*
                                         * get the first entry in the extensions
                                         * list on the inode record
                                         */
                                        this_ext = this_inorec->ext_rec;

                                        while ((cdi_rc == FSCK_OK)
                                               && (this_ext != NULL)) {
                                                /* there may be more parents */
                                                if (this_ext->ext_type ==
                                                    parent_extension) {
                                                        /* a parent */
                                                        cdi_rc =
                                                            get_inorecptr
                                                            (aggregate_inode,
                                                             alloc_ifnull,
                                                             this_ext->inonum,
                                                             &that_inorec);
                                                        if ((cdi_rc == FSCK_OK)
                                                            && (!that_inorec->
                                                                selected_to_rls))
                                                        {
                                                                /* parent isn't marked for release (yet) */
                                                                dup_parent_detected
                                                                    = 0;
                                                                if (this_ext->
                                                                    inonum ==
                                                                    this_inorec->
                                                                    parent_inonum)
                                                                {
                                                                        dup_parent_detected
                                                                            =
                                                                            -1;
                                                                } else {
                                                                        /* need to check for dups in rest of list
 */
                                                                        that_ext
                                                                            =
```

```
630                                                                                 this_ext−>
631                                                                                 next;
632                                                                         while ((!dup_parent_detected) && (that_ex
    t != NULL)) {
633                                                                                 if (that_ext−>ext_type == parent_
    extension)
634                                                                                         /* another parent extensi
    on */
635                                                                                 if (this_ext−>inonum == t
    hat_ext−>inonum) {
636                                                                                         dup_parent_detect
    ed
637                                                                                             =
638                                                                                             −1;
639                                                                                 }
640                                                                                 that_ext
641                                                                                     =
642                                                                                     that_ext−>
643                                                                                     next;
644                                                                         }
645                                                                 }
646                                                                 if (dup_parent_detected) {
647                                                                         /*
648                                                                          * mark the parent’s inode record for rel
    ease
649                                                                          */
650                                                                         that_inorec−>
651                                                                             selected_to_rls
652                                                                             = 1;
653                                                                         /*
654                                                                          * notify the user that the directory is
    bad
655                                                                          */
656                                                                         msgprms
657                                                                             [0]
658                                                                             =
659                                                                             message_parm_0;
660                                                                         msgprmidx
661                                                                             [0]
662                                                                             =
663                                                                             fsck_directory;
664                                                                         msgprms
665                                                                             [1]
666                                                                             =
667                                                                             message_parm_1;
668                                                                         msgprmidx
669                                                                             [1]
670                                                                             =
671                                                                             msg_info_ptr−>
672                                                                             msg_inopfx;
673                                                                         sprintf
674                                                                             (message_parm_2,
675                                                                              "%d",
676                                                                              this_ext−>
677                                                                              inonum);
678                                                                         msgprms
679                                                                             [2]
680                                                                             =
681                                                                             message_parm_2;
682                                                                         msgprmidx
683                                                                             [2]
684                                                                             = 0;
685                                                                         sprintf
686                                                                             (message_parm_3,
687                                                                              "%d",
688                                                                              37);
689                                                                         msgprms
690                                                                             [3]
691                                                                             =
692                                                                             message_parm_3;
693                                                                         msgprmidx
694                                                                             [3]
695                                                                             = 0;
696                                                                         fsck_send_msg
697                                                                             (fsck_BADKEYS,
698                                                                              4);
699                                                                 }
700                                                         }
701                                                 }
702                                                 this_ext = this_ext−>next;
703                                         }
704                                 }
705                         }
706                 }
707                 if (cdi_rc == FSCK_OK) {
708                         cdi_rc =
709                             get_inorecptr_next(aggregate_inode, &ino_idx,
710                                                &this_inorec);
711                 }
712         }
```

```
713
714         return (cdi_rc);
715 }
716
717 /**************************************************************************
718  * NAME: check_link_counts
719  *
720  * FUNCTION:  Count links from child directories to their parents.
721  *
722  *            Verify that the link count stored in each in−use inode
723  *            matches the number of links fsck observed for the inode.
724  *
725  * RETURNS:
726  *      success: FSCK_OK
727  *      failure: something else
728  */
729 int check_link_counts()
730 {
731         int clc_rc = FSCK_OK;
732         uint32_t ino_idx;
733         int num_parents;
734         int invalid_count_seen = 0;
735         int low_stored_count_seen = 0;
736         struct fsck_inode_ext_record *this_ext;
737         struct fsck_inode_record *this_inorec;
738         struct fsck_inode_record *parent_inorec;
739         int done_looking = 0;
740         int aggregate_inode = 0;
741         int alloc_ifnull = 0;
742
743         struct fsck_ino_msg_info ino_msg_info;
744         struct fsck_ino_msg_info *msg_info_ptr;
745
746         msg_info_ptr = &ino_msg_info;
747         /* all fileset owned */
748         msg_info_ptr->msg_inopfx = fsck_fset_inode;
749
750         /*
751          * count links from child directories to their parents
752          *
753          * (These can't be counted when the parent−child relationship
754          * is observed because that observation occurs while processing
755          * the parent and until the child is processed we don't know
756          * whether the child is a directory or not.)
757          */
758         clc_rc = get_inorecptr_first(aggregate_inode, &ino_idx, &this_inorec);
759
760         while ((clc_rc == FSCK_OK) && (this_inorec != NULL)) {
761                 if ((this_inorec->in_use) && (!this_inorec->selected_to_rls)
762                     && (!this_inorec->ignore_alloc_blks)
763                     && (this_inorec->inode_type == directory_inode)) {
764                         /* inode is in use, not being released, and is type directory */
765                         /* for the self entry */
766                         this_inorec->link_count++;
767                         if ((this_inorec->parent_inonum == ROOT_I)
768                             && (agg_recptr->rootdir_rebuilt)) {
769                                 /*
770                                  * special case: if the parent is root and root was
771                                  * rebuilt, then don't increment parent
772                                  */
773                                 if (this_inorec->inonum == ROOT_I) {
774                                         /*
775                                          * special case: if this IS the root, then it's
776                                          * link from itself to itself DOES count
777                                          */
778                                         this_inorec->link_count++;
779                                 }
780                         } else if (this_inorec->parent_inonum != 0) {
781                                 /* not an orphan */
782                                 clc_rc =
783                                     get_inorecptr(aggregate_inode, alloc_ifnull,
784                                                   this_inorec->parent_inonum,
785                                                   &parent_inorec);
786                                 if ((clc_rc != FSCK_OK)
787                                     || (parent_inorec == NULL)) {
788                                         clc_rc = FSCK_INTERNAL_ERROR_13;
789                                         sprintf(message_parm_0, "%d", clc_rc);
790                                         msgprms[0] = message_parm_0;
791                                         msgprmidx[0] = 0;
792                                         sprintf(message_parm_1, "%d", ino_idx);
793                                         msgprms[1] = message_parm_1;
794                                         msgprmidx[1] = 0;
795                                         sprintf(message_parm_2, "%d",
796                                                 this_inorec->parent_inonum);
797                                         msgprms[2] = message_parm_2;
798                                         msgprmidx[2] = 0;
799                                         sprintf(message_parm_3, "%d", 0);
800                                         msgprms[3] = message_parm_3;
801                                         msgprmidx[3] = 0;
802                                         fsck_send_msg(fsck_INTERNALERROR, 4);
```

```
803                                     } else {
804                                             /* handle the first (and usually the only) parent. */
805                                             parent_inorec->link_count++;
806                                     }
807                                     if ((clc_rc == FSCK_OK)
808                                         && (this_inorec->ext_rec != NULL)) {
809                                             /* there might be more parents */
810                                             num_parents = parent_count(this_inorec);
811                                             if (num_parents > 1) {
812                                                     /* directory with illegal links */
813                                                     this_inorec->unxpctd_prnts = 1;
814                                                     agg_recptr->corrections_needed =
815                                                       1;
816                                                     /*
817                                                      * Create an extension record for the parent inode
818                                                      * number now stored in the child inode record.
819                                                      * When we traverse the aggregate on-disk we'll copy
820                                                      * the stored value into this field of the inode record
821                                                      * for use when displaying paths to the inode.
822                                                      */
823                                                     clc_rc =
824                                                         get_inode_extension
825                                                         (&this_ext);
826                                                     if (clc_rc == FSCK_OK) {
827                                                             /* got extension record */
828                                                             this_ext->ext_type =
829                                                                 parent_extension;
830                                                             this_ext->inonum =
831                                                                 this_inorec->
832                                                                 parent_inonum;
833                                                             this_ext->next =
834                                                                 this_inorec->
835                                                                 ext_rec;
836                                                             this_inorec->ext_rec =
837                                                                 this_ext;
838                                                             this_inorec->
839                                                                 parent_inonum = 0;
840                                                             /* already counted the first
841                                                              * one, back when it was in the
842                                                              * workspace inode record itself
843                                                              */
844                                                             this_ext =
845                                                                 this_ext->next;
846                                                             while ((clc_rc ==
847                                                                     FSCK_OK)
848                                                                    && (this_ext !=
849                                                                        NULL)) {
850                                                                     /* exten records to check */
851                                                                     if (this_ext->
852                                                                         ext_type ==
853                                                                         parent_extension)
854                                                                     {
855                                                                             clc_rc =
856                                                                                 get_inorecptr
857                                                                                 (aggregate_inode,
858                                                                                  alloc_ifnull,
859                                                                                  this_ext->
860                                                                                  inonum,
861                                                                                  &parent_inorec);
862                                                                             if ((clc_rc != FSCK_OK)
863                                                                                 ||
864                                                                                 (parent_inorec
865                                                                                  ==
866                                                                                  NULL))
867                                                                             {
868                                                                                     clc_rc
869                                                                                         =
870                                                                                         FSCK_INTERNAL_ERROR_14;
871                                                                                     sprintf
872                                                                                         (message_parm_0,
873                                                                                          "%d",
874                                                                                          clc_rc);
875                                                                                     msgprms
876                                                                                         [0]
877                                                                                         =
878                                                                                         message_parm_0;
879                                                                                     msgprmidx
880                                                                                         [0]
881                                                                                         =
882                                                                                         0;
883                                                                                     sprintf
884                                                                                         (message_parm_1,
885                                                                                          "%d",
886                                                                                          ino_idx);
887                                                                                     msgprms
888                                                                                         [1]
889                                                                                         =
890                                                                                         message_parm_1;
891                                                                                     msgprmidx
892                                                                                         [1]
```

```
                                                                                =
                                                                                0;
                                                                            sprintf
                                                                                (message_parm_2,
                                                                                 "%d",
                                                                                 this_ext->
                                                                                 inonum);
                                                                            msgprms
                                                                                [2]
                                                                                =
                                                                                message_parm_2;
                                                                            msgprmidx
                                                                                [2]
                                                                                =
                                                                                0;
                                                                            sprintf
                                                                                (message_parm_3,
                                                                                 "%d",
                                                                                 0);
                                                                            msgprms
                                                                                [3]
                                                                                =
                                                                                message_parm_3;
                                                                            msgprmidx
                                                                                [3]
                                                                                =
                                                                                0;
                                                                            fsck_send_msg
                                                                                (fsck_INTERNALERROR,
                                                                                 4);
                                                                    } else {
                                                                            parent_inorec->
                                                                                link_count++;
                                                                    }
                                                            }
                                                            this_ext =
                                                                this_ext->
                                                                next;
                                                    }
                                            }
                                    }
                            }
                    }
                    if (clc_rc == FSCK_OK) {
                            clc_rc =
                                get_inorecptr_next(aggregate_inode, &ino_idx,
                                                   &this_inorec);
                    }
            }

            /*
             * verify that stored link counts match observed link counts
             *
             * We have added each observed link and subtracted the stored
             * count.  If the stored count is correct the result is 0.
             */
            if (clc_rc == FSCK_OK) {
                    clc_rc =
                        get_inorecptr_first(aggregate_inode, &ino_idx,
                                            &this_inorec);

                    while ((clc_rc == FSCK_OK) && (this_inorec != NULL)
                           && (!done_looking)) {

                            if ((this_inorec->in_use)
                                && (!this_inorec->selected_to_rls)
                                && (!this_inorec->ignore_alloc_blks)) {
                                /* inode is in use and not being released */
                                if (this_inorec->link_count != 0) {
                                        /* stored link count doesn't match fsck's observations */
                                        if (this_inorec->parent_inonum == 0) {
                                                /* inode is an orphan */
                                                this_inorec->crrct_link_count =
                                                    1;
                                        } else {
                                                /* not an orphan */
                                                this_inorec->crrct_link_count =
                                                    1;
                                                if (this_inorec->link_count > 0) {
                                                        low_stored_count_seen =
                                                            1;
                                                }
                                                invalid_count_seen = 1;

                                                msgprms[0] = message_parm_0;
                                                msgprmidx[0] =
                                                    msg_info_ptr->msg_inopfx;
                                                sprintf(message_parm_1, "%d",
                                                        ino_idx);
```

```
                                                msgprms[1] = message_parm_1;
                                                msgprmidx[1] = 0;
                                                fsck_send_msg(fsck_BADINOLKCT,
                                                              2);
                                        }
                                }
                        }
                        if (clc_rc == FSCK_OK) {
                                clc_rc =
                                    get_inorecptr_next(aggregate_inode,
                                                       &ino_idx, &this_inorec);
                        }
                }

                if ((clc_rc == FSCK_OK) && (invalid_count_seen)) {

                        if (agg_recptr->processing_readwrite) {
                                agg_recptr->corrections_approved = 1;
                                fsck_send_msg(fsck_WILLFIXLINKCTS, 0);
                        } else {
                                /* no write access */
                                if (low_stored_count_seen) {
                                        agg_recptr->ag_dirty = 1;
                                }
                                /*
                                 * reset all link counts (in the fsck workspace) to
                                 * zero so that we won't accidentally correct them
                                 * while doing link count adjustments.
                                 *
                                 * (Link count adjustments are side effects of approved
                                 * repairs.  For example, if a directory inode is
                                 * released, the link count of its parent directory
                                 * is decremented.)
                                 */
                                clc_rc =
                                    get_inorecptr_first(aggregate_inode,
                                                        &ino_idx, &this_inorec);

                                while ((clc_rc == FSCK_OK)
                                       && (this_inorec != NULL)) {

                                        if (this_inorec->in_use) {
                                                this_inorec->crrct_link_count =
                                                    0;
                                                this_inorec->link_count = 0;
                                        }
                                        clc_rc =
                                            get_inorecptr_next(aggregate_inode,
                                                               &ino_idx,
                                                               &this_inorec);
                                }
                                fsck_send_msg(fsck_BADLINKCTS, 0);
                        }
                }
        }
        return (clc_rc);
}

/*****************************************************************************
 * NAME: reset_parents
 *
 * FUNCTION: Adjust the fsck notations about the inode's parent(s) if
 *           the parent(s) are corrupt or approved for release.
 *
 * PARAMETERS:
 *      ino_recptr  − input − pointer to an fsck record describing the inode
 *      ino_idx     − input − ordinal number of the inode
 *
 * RETURNS:
 *      success: FSCK_OK
 *      failure: something else
 */
int reset_parents(struct fsck_inode_record *ino_recptr, uint32_t ino_idx)
{
        int resps_rc = FSCK_OK;
        struct fsck_inode_ext_record *this_ext;
        struct fsck_inode_ext_record *rest_of_list;
        int parent_count = 0;
        uint32_t stored_parent_inonum = 0;
        struct fsck_inode_record *parent_inorecptr;
        int aggregate_inode = 0;
        int alloc_ifnull = 0;
        /*
         * if this is a directory with illegal hard links the inode
         * number in the fsck inode record is the one stored in the
         * inode on disk.
         */
        if ((ino_recptr->inode_type == directory_inode)
            && (ino_recptr->unxpctd_prnts)) {
                /* dir with multiple parents */
```

```
1073                    /*
1074                     * Save the value stored in the inode record and then clear it.
1075                     */
1076                    stored_parent_inonum = ino_recptr->parent_inonum;
1077                    ino_recptr->parent_inonum = 0;
1078            } else {
1079                    /* not a dir with multiple parents */
1080                    /*
1081                     * the 1st parent observed is in the inode record.
1082                     * Any others are in extension records.
1083                     */
1084                    resps_rc = get_inorecptr(aggregate_inode, alloc_ifnull,
1085                                             ino_recptr->parent_inonum,
1086                                             &parent_inorecptr);
1087                    if ((resps_rc == FSCK_OK) && (parent_inorecptr == NULL)) {
1088                            resps_rc = FSCK_INTERNAL_ERROR_15;
1089                            sprintf(message_parm_0, "%d", resps_rc);
1090                            msgprms[0] = message_parm_0;
1091                            msgprmidx[0] = 0;
1092                            sprintf(message_parm_1, "%d", ino_idx);
1093                            msgprms[1] = message_parm_1;
1094                            msgprmidx[1] = 0;
1095                            sprintf(message_parm_2, "%d",
1096                                    ino_recptr->parent_inonum);
1097                            msgprms[2] = message_parm_2;
1098                            msgprmidx[2] = 0;
1099                            sprintf(message_parm_3, "%d", 0);
1100                            msgprms[3] = message_parm_3;
1101                            msgprmidx[3] = 0;
1102                            fsck_send_msg(fsck_INTERNALERROR, 4);
1103
1104                    } else if (resps_rc == FSCK_OK) {
1105                            if ((ino_recptr->parent_inonum == ROOT_I)
1106                                && (agg_recptr->rootdir_rebuilt)) {
1107                                    /*
1108                                     * special case: if the parent is root and root
1109                                     * was rebuilt, then this is an orphan
1110                                     */
1111                                    ino_recptr->parent_inonum = 0;
1112                                    ino_recptr->link_count--;
1113                            } else if ((!parent_inorecptr->selected_to_rls)
1114                                       && (!parent_inorecptr->ignore_alloc_blks)) {
1115                                    /*
1116                                     * keeping this parent and haven't found the
1117                                     * tree to be corrupt
1118                                     */
1119                                    parent_count++;
1120
1121                            } else {
1122                                    /* releasing this parent */
1123                                    ino_recptr->parent_inonum = 0;
1124                                    ino_recptr->link_count--;
1125                            }
1126                    }
1127            }
1128            /*
1129             * detach the extensions list from the inode record
1130             */
1131            this_ext = ino_recptr->ext_rec;
1132            ino_recptr->ext_rec = NULL;
1133            while ((resps_rc == FSCK_OK) && (this_ext != NULL)) {
1134                    /* there may be more parents */
1135                    rest_of_list = this_ext->next;
1136                    if (this_ext->ext_type != parent_extension) {
1137                            /* not a parent */
1138                            this_ext->next = ino_recptr->ext_rec;
1139                            ino_recptr->ext_rec = this_ext;
1140                    } else {
1141                            /* parent extension */
1142                            resps_rc = get_inorecptr(aggregate_inode, alloc_ifnull,
1143                                                     this_ext->inonum,
1144                                                     &parent_inorecptr);
1145                            if ((resps_rc == FSCK_OK) && (parent_inorecptr == NULL)) {
1146                                    resps_rc = FSCK_INTERNAL_ERROR_16;
1147                                    sprintf(message_parm_0, "%d", resps_rc);
1148                                    msgprms[0] = message_parm_0;
1149                                    msgprmidx[0] = 0;
1150                                    sprintf(message_parm_1, "%d", ino_idx);
1151                                    msgprms[1] = message_parm_1;
1152                                    msgprmidx[1] = 0;
1153                                    sprintf(message_parm_2, "%d", this_ext->inonum);
1154                                    msgprms[2] = message_parm_2;
1155                                    msgprmidx[2] = 0;
1156                                    sprintf(message_parm_3, "%d", 0);
1157                                    msgprms[3] = message_parm_3;
1158                                    msgprmidx[3] = 0;
1159                                    fsck_send_msg(fsck_INTERNALERROR, 4);
1160
1161                            } else if (resps_rc == FSCK_OK) {
1162                                    if ((ino_recptr->parent_inonum == ROOT_I)
```

```
1163                                    && (agg_recptr->rootdir_rebuilt)) {
1164                                        /*
1165                                         * special case: if the parent is root and
1166                                         * root was rebuilt, then this is an orphan
1167                                         */
1168                                        release_inode_extension(this_ext);
1169                                        ino_recptr->link_count--;
1170
1171                                } else if ((!parent_inorecptr->selected_to_rls)
1172                                           && (!parent_inorecptr->
1173                                               ignore_alloc_blks)) {
1174                                        /* keeping this parent */
1175                                        parent_count++;
1176                                        if (ino_recptr->parent_inonum == 0) {
1177                                                ino_recptr->parent_inonum =
1178                                                    this_ext->inonum;
1179                                                release_inode_extension
1180                                                    (this_ext);
1181                                        } else {
1182                                                /* put it back on the list */
1183                                                this_ext->next =
1184                                                    ino_recptr->ext_rec;
1185                                                ino_recptr->ext_rec = this_ext;
1186                                        }
1187
1188                                } else {
1189                                        /* releasing this parent */
1190                                        release_inode_extension(this_ext);
1191                                        ino_recptr->link_count--;
1192                                }
1193                        }
1194                }
1195                this_ext = rest_of_list;
1196        }
1197        /*
1198         * at this point, if there is at least 1 observed parent which
1199         * is not being released, then a parent inode number is stored in
1200         * the inode record and any other parents are described in extension
1201         * records.
1202         *
1203         * if this is not a directory inode, we're done.
1204         *
1205         * if this is a directory inode, need to recheck for illegal hard
1206         * links and incorrect parent inode entry.
1207         */
1208        if ((resps_rc == FSCK_OK)
1209            && (ino_recptr->inode_type == directory_inode)) {
1210                /* a directory */
1211                if (parent_count == 1) {
1212                        /* 1 parent now */
1213                        if (ino_recptr->unxpctd_prnts) {
1214                                /* entered with multiple links */
1215                                /* reset flag */
1216                                ino_recptr->unxpctd_prnts = 0;
1217                                if (ino_recptr->parent_inonum !=
1218                                    stored_parent_inonum) {
1219                                        /*
1220                                         * Remaining parent doesn't match the one
1221                                         * the on-device inode says owns it.
1222                                         */
1223                                        ino_recptr->crrct_prnt_inonum = 1;
1224                                        agg_recptr->corrections_needed = 1;
1225                                        agg_recptr->corrections_approved = 1;
1226                                }
1227                        }
1228                } else if (parent_count == 0) {
1229                        /* no parents now */
1230                        ino_recptr->crrct_prnt_inonum = 0;
1231                        ino_recptr->unxpctd_prnts = 0;
1232                } else {
1233                        /* multiple parents still */
1234                        ino_recptr->unxpctd_prnts = 1;
1235                        agg_recptr->corrections_needed = 1;
1236                        resps_rc = get_inode_extension(&this_ext);
1237                        if (resps_rc == FSCK_OK) {
1238                                this_ext->ext_type = parent_extension;
1239                                this_ext->inonum = ino_recptr->parent_inonum;
1240                                this_ext->next = ino_recptr->ext_rec;
1241                                ino_recptr->ext_rec = this_ext;
1242                                ino_recptr->parent_inonum =
1243                                    stored_parent_inonum;
1244                        }
1245                }
1246        }
1247        return (resps_rc);
1248 }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_FSCKPFS
#define H_FSCKPFS

#define fsck_READ  1
#define fsck_WRITE 2

#endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_PUFS_CHKDSK
#define H_PUFS_CHKDSK

#include "xfsck.h"

enum xchkdsk_options {
        UFS_CHKDSK_LEVEL0,
        UFS_CHKDSK_LEVEL1,
        UFS_CHKDSK_LEVEL2,
        UFS_CHKDSK_LEVEL3,
        UFS_CHKDSK_IFDIRTY,
        UFS_CHKDSK_VERBOSE,
        UFS_CHKDSK_DEBUG,
        UFS_CHKDSK_CLRBDBLKLST,
        UFS_CHKDSK_SKIPLOGREDO,
        UFS_CHKDSK_BLOCK_NUMBER,
        UFS_CHKDSK_INODE_NUMBER,
        UFS_CHKDSK_FILENAME,
        UFS_CHKDSK_OPTIONS
};

#endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_XFSCK
#define H_XFSCK

/*
   −−−−−−−−−−−−−−−−−−−−−−−−− system includes  −−−−−−−−−−−−−−−−−−−−−−−−−
*/

#include <stdarg.h>
#include <stddef.h>

#ifndef _JFS_FSCKDIRE
#include <stdio.h>
#endif

#include <string.h>
#include <stdlib.h>

/*
   −−−−−−−−−−−−−−−−−−−−−−−−−−− JFS includes  −−−−−−−−−−−−−−−−−−−−−−−−−−−
*/

#include "jfs_types.h"
#include "jfs_dinode.h"
#include "jfs_dmap.h"
#include "jfs_filsys.h"
#include "jfs_imap.h"
#include "jfs_superblock.h"
#include "fsck_base.h"
#include "fsckcbbl.h"

/*
 * jfs library routine includes
 */
#include "libjufs.h"
#include "devices.h"
#include "diskmap.h"
#include "inode.h"
#include "logform.h"
#include "logredo.h"
#include "message.h"
#include "super.h"

/* Stuff for handling extended attributes. */
#ifndef OS2

struct FEA {                    /* fea */
        uint8_t fEA;            /* flags                                */
        uint8_t cbName;         /* name length not including NULL */
        uint16_t cbValue;       /* value length */
};

/* flags for FEA.fEA */
#define FEA_NEEDEA         0x80 /* need EA bit */

struct FEALIST {                /* feal */
        uint32_t cbList;        /* total bytes of structure including full list */
        struct FEA list[1];     /* variable length FEA structures */
};

#define ERROR_EA_LIST_INCONSISTENT  255

#endif

extern int jfs_ValidateFEAList(struct FEALIST *, unsigned long *);

/* ***** IMPORTANT ***** IMPORTANT ***** IMPORTANT ***** IMPORTANT *****
 *
 * fsck_first_msgid
 *      MUST be set to the first message id for fsck, according
 *      to jfs.txt.  Will be used to locate the text for any
 *      message which is displayed in the local language.
 *
 * fsck_highest_msgid_defined
```

```
 *      MUST be maintained in synch with the
 *      message id constants (defined in fsckmsgc.h) since the
 *      message text array and the message attributes array (declared
 *      in fsckmsgp.h) are both dimensioned using it.
 *
 * ***** IMPORTANT ***** IMPORTANT ***** IMPORTANT ***** IMPORTANT ***** */

#define fsck_msgid_offset           50
#define fsck_highest_msgid_defined 599

#define MAXPARMLEN 64

#define fscklog_var_text 1
#define fscklog_literal  2

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * The following are used to access the columns of
  * MsgProtocol[][], which is defined in fsckmsgp.h
  *
  */
#define MP_MSGLVL     0
#define MP_MSGFILE    1

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  *
  * The following are the possible values effective fsck
  * messaging level and for implied response level.
  *
  */

/*
 * special cases (must not match any other constants in the table)
 */
#define fsck_hrtbt     130
#define fsck_txtins    131

/*
 * The lowest messaging level (dictated by input parms) at which
 * the message is displayed.
 */
#define fsck_quiet       2
#define fsck_verbose     8
#define fsck_debug      32

/*
 * The message file to use for local language lookup
 *
 *   These are determined by the array message_file_name[]
 *   in messages.c
 */
#define no_msgfile      −1
#define jfs_msgfile      1

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
    * The following are used for reporting storage allocation
    * failures.
  */

#ifndef _JFS_FSCKDIRE

void report_dynstg_error(void);

#define  dynstg_unknown          0

/* the actions */

#define dynstg_allocation        1
#define dynstg_initialization    2

/* the objects */

#define dynstg_unknown_object    0
#define dynstg_blkmap            1
#define dynstg_blkmap_buf        2
#define dynstg_blkmap_hdr        3
#define dynstg_inomap            4
#define dynstg_fer               5
#define dynstg_wspext            6
#define dynstg_pathbuffer        7
#define dynstg_inoextrec         8
#define dynstg_inorec            9
#define dynstg_dtreeQ_elem      10
#define dynstg_treeQ_elem       11
#define dynstg_ait_map          12
#define dynstg_fsit_map         13
#define dynstg_dupall_blkrec    14
#define dynstg_dupall_inorec    15
#define dynstg_agg_agtbl        16
#define dynstg_agg_iagtbl       17
```

```
#define dynstg_fs_agtbl         18
#define dynstg_fs_iagtbl        19
#define dynstg_iobufs           20
#define dynstg_eaiobuf          21
#define dynstg_tmpinoiobuf      22
#define dynstg_recondnodebuf    23
#define dynstg_xtreebuf         24
#define dynstg_xtreepagebuf     25
#define dynstg_treeStack_elem   26
#define dynstg_fsckcbblbuf1     27
#define dynstg_fsckcbblbuf2     28
#define dynstg_inotbl           29
#define dynstg_inoexttbl        30
#define dynstg_fsit_iagtbl      31
#define dynstg_fsit_inoexttbl   32
#define dynstg_fsit_inotbl      33
#define dynstg_ait_inotbl       34

#endif                          /* ! _JFS_FSCKDIRE */

 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
  * The following are the possible values for current_fsck_phase
  * which is declared in xchkdsk.c
  *
  */
#define fsck_entry         16   /* x10 => entry linkage  */
#define fsck_phase_initial 17   /* x11 => initialization */
#define fsck_phase0         0
#define fsck_phase1         1
#define fsck_phase2         2
#define fsck_phase3         3
#define fsck_phase4         4
#define fsck_phase5         5
#define fsck_phase6         6
#define fsck_phase7         7
#define fsck_phase8         8
#define fsck_phase9         9
#define fsck_phase_final   32   /* x20 => cleanup        */
#define fsck_exit          33   /* x21 => exit linkage   */

/* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
 *
 * used during blockmap verify/rebuild
 *
 */
struct fsck_bmap_record {
        char eyecatcher[8];
        int64_t total_blocks;
        int64_t free_blocks;
        int8_t ctl_fctl_error;
        int8_t ctl_other_error;
        char rsvd0[2];

        char bmpctlinf_eyecatcher[8];
        struct dbmap *bmpctl_bufptr;
        int64_t bmpctl_agg_fsblk_offset;
        char rsvd1[4];

        char AGinf_eyecatcher[8];
        int64_t *AGFree_tbl;
        int8_t AGActive[MAXAG];

        char dmapinf_eyecatcher[8];
        uint32_t dmappg_count;
        uint32_t dmappg_ordno;
        uint32_t dmappg_idx;
        int8_t *dmap_wsp_stree;
        int8_t *dmap_wsp_sleafs;
        struct dmap *dmap_bufptr;
        int64_t dmap_agg_offset;
        int64_t dmap_1stblk;
        int8_t dmap_pmap_error;
        int8_t dmap_slfv_error;
        int8_t dmap_slnv_error;
        int8_t dmap_other_error;
        char rsvd2[4];

        char L0inf_eyecatcher[8];
        uint32_t L0pg_count;
        uint32_t L0pg_ordno;
        uint32_t L0pg_idx;
        int8_t *L0_wsp_stree;
        int8_t *L0_wsp_sleafs;
        struct dmapctl *L0_bufptr;
        int64_t L0pg_agg_offset;
        int64_t L0pg_1stblk;
        int8_t L0_rsvd;
        int8_t L0pg_slfv_error;
        int8_t L0pg_slnv_error;
        int8_t L0pg_other_error;
```

```
271         char rsvd3[4];
272
273         char L1inf_eyecatcher[8];
274         uint32_t L1pg_count;
275         uint32_t L1pg_ordno;
276         uint32_t L1pg_idx;
277         int8_t *L1_wsp_stree;
278         int8_t *L1_wsp_sleafs;
279         struct dmapctl *L1_bufptr;
280         int64_t L1pg_agg_offset;
281         int64_t L1pg_1stblk;
282         int8_t L1_rsvd;
283         int8_t L1pg_slfv_error;
284         int8_t L1pg_slnv_error;
285         int8_t L1pg_other_error;
286         char rsvd4[4];
287
288         char L2inf_eyecatcher[8];
289         uint32_t L2pg_count;
290         int8_t *L2_wsp_stree;
291         int8_t *L2_wsp_sleafs;
292         struct dmapctl *L2_bufptr;
293         int64_t L2pg_agg_offset;
294         int64_t L2pg_1stblk;
295         int8_t L2pg_slfv_error;
296         int8_t L2pg_slnv_error;
297         int8_t L2pg_other_error;
298         char rsvd5[5];
299 };
300
301 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
302  *
303  * unicharacter name structure
304  */
305 struct uniname {
306         uint32_t len_in_UniChars;
307         UniChar name_in_UniChars[256];
308 };
309
310 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
311  *
312  * used to pass message inserts from one routine to another economically
313  */
314 struct fsck_ino_msg_info {
315         uint32_t msg_inonum;
316         int msg_inopfx;
317         int msg_inotyp;
318         int msg_dxdtyp;
319 };
320
321 struct fsck_imap_msg_info {
322         int32_t msg_iagnum;
323         int32_t msg_agnum;
324         int msg_mapowner;
325 };
326
327 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
328  *
329  * used to specify what action should be performed by routing
330  * process_extent
331  */
332 #define FSCK_RECORD                 1
333 #define FSCK_RECORD_DUPCHECK        2
334 #define FSCK_UNRECORD               3
335 #define FSCK_QUERY                  4
336 #define FSCK_GET_OBJNAME            5
337 #define FSCK_FSIM_RECORD_DUPCHECK   6
338 #define FSCK_FSIM_UNRECORD          7
339 #define FSCK_FSIM_QUERY             8
340
341  /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
342   *
343   * The following are the values which may be displayed in
344   * message fsck_INOINLINECONFLICT
345   *
346   */
347 #define fsck_longdata_and_otherinline  1
348
349 /* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
350  *
351  * The following define exit codes passed back by fsck.jfs
352  * and return codes passed internally in fsck.jfs.
353  *
354  */
355
356 /* exit codes */
357
358 #define FSCK_OK                   0
359 #define FSCK_CORRECTED            1
360 #define FSCK_REBOOT               2
```

```
#define FSCK_ERRORS_UNCORRECTED    4
#define FSCK_OP_ERROR              8
#define FSCK_USAGE_ERROR          16

/* informational return codes */

#define FSCK_FAILED             −00001

#define FSCK_AGFS_INOBAD          10001
#define FSCK_AGGAITINOBAD         10002
#define FSCK_AGGFSINOBAD          10003
#define FSCK_BADAGFSSIZ           10004
#define FSCK_BADBLKCTTTL          10005
#define FSCK_BADEADESCRIPTOR      10006
#define FSCK_BADMDDATA            10007
#define FSCK_BADMDDATAEXT         10008
#define FSCK_BADMDDATAIDX         10009
#define FSCK_BADMDDATAINLN        10010
#define FSCK_BADREAD_FBLKMP       10011
#define FSCK_BADREAD_FSCKLOG      10012
#define FSCK_BADSBAGSIZ           10013
#define FSCK_BADSBFJLA            10014
#define FSCK_BADSBFJLL            10015
#define FSCK_BADSBFSSIZ           10016
#define FSCK_BADSBFWSA            10017
#define FSCK_BADSBFWSL            10018
#define FSCK_BADSBFWSL1           10019
#define FSCK_BADSBMGC             10020
#define FSCK_BADSBVRSN            10021
#define FSCK_BADSBOTHR1           10022
#define FSCK_BADSBOTHR2           10023
#define FSCK_BADSBOTHR3           10024
#define FSCK_BADSBOTHR4           10025
#define FSCK_BADSBOTHR5           10026
#define FSCK_BADSBOTHR6           10027
#define FSCK_BADSBOTHR7           10028
#define FSCK_BADSBOTHR8           10029
#define FSCK_BADSBOTHR9           10030
#define FSCK_BADSBOTHR10          10031
#define FSCK_BADSBOTHR11          10032
#define FSCK_BADSBOTHR12          10033
#define FSCK_BADSBOTHR13          10034
#define FSCK_BADWRITE_FSCKLOG     10035
#define FSCK_BADWRITE_FBLKMP      10036
#define FSCK_BBINOBAD             10037
#define FSCK_BLSIZLTLVBLSIZ       10038
#define FSCK_BMINOBAD             10039
#define FSCK_CANT_ALLOC_INOREC    10040
#define FSCK_CANT_ALLOC_LSFN      10041
#define FSCK_CANT_EXTEND_ROOTDIR  10042
#define FSCK_CANTREADEAITEXT1     10043
#define FSCK_CANTREADAITEXT1      10044
#define FSCK_CANTREADAITEXT2      10045
#define FSCK_CANTREADAITEXT3      10046
#define FSCK_CANTREADAITEXT4      10047
#define FSCK_CANTREADAGGFSINO     10048
#define FSCK_CANTREADBBINO        10049
#define FSCK_CANTREADBMINO        10050
#define FSCK_CANTREADEA           10051
#define FSCK_CANTREADFSEXT        10052
#define FSCK_CANTREADFSRTDR       10053
#define FSCK_CANTREADLOGINO       10054
#define FSCK_CANTREADRECONDNODE   10055
#define FSCK_CANTREADRECONDNODE1  10056
#define FSCK_CANTREADSELFINO      10057
#define FSCK_CANTWRITRECONDNODE   10058
#define FSCK_CANTWRITRECONDNODE1  10059
#define FSCK_DUPMDBLKREF          10060
#define FSCK_FSETEXTBAD           10061
#define FSCK_FSRTDRBAD            10062
#define FSCK_IAGNOOOAGGBOUNDS     10063
#define FSCK_IAGNOOOFSETBOUNDS    10064
#define FSCK_INOEXTNOTALLOC       10065
#define FSCK_INOINLINECONFLICT1   10066
#define FSCK_INOINLINECONFLICT2   10067
#define FSCK_INOINLINECONFLICT3   10068
#define FSCK_INOINLINECONFLICT4   10069
#define FSCK_INOINLINECONFLICT5   10070
#define FSCK_LOGINOBAD            10071
#define FSCK_RIBADTREE            10072
#define FSCK_RIDATAERROR          10073
#define FSCK_RINOTDIR             10074
#define FSCK_RIUNALLOC            10075
#define FSCK_SELFINOBAD           10076
#define FSCK_INSUFDSTG4RECON      10077
#define FSCK_INSUFDSTG4RECON1     10078
#define FSCK_BLKSNOTAVAILABLE     10079
#define FSCK_BADREADTARGET        10080
#define FSCK_BADREADTARGET1       10081
#define FSCK_BADREADTARGET2       10082
```

```
451 #define FSCK_ENOMEMBDBLK1         10083
452 #define FSCK_ENOMEMBDBLK2         10084
453 #define FSCK_ENOMEMBDBLK3         10085
454 #define FSCK_ENOMEMBDBLK4         10086
455 #define FSCK_PARENTNULLIFIED      10087
456 #define FSCK_IOTARGETINJRNLLOG    10088
457
458 /* fatal condition return codes */
459
460 #define FSCK_FAILED_SEEK             −10001
461 #define FSCK_FAILED_BADSEEK          −10002
462
463 #define FSCK_FAILED_NODE_BADFLUSH    −10003
464 #define FSCK_FAILED_NODE_FLUSH       −10004
465 #define FSCK_FAILED_BADREAD_NODE     −10005
466 #define FSCK_FAILED_BADREAD_NODE1    −10006
467 #define FSCK_FAILED_READ_NODE        −10007
468 #define FSCK_FAILED_READ_NODE2       −10008
469 #define FSCK_FAILED_READ_NODE3       −10009
470 #define FSCK_FAILED_READ_NODE4       −10010
471
472 #define FSCK_FAILED_BADREAD_DNODE    −10011
473 #define FSCK_FAILED_READ_DNODE       −10012
474
475 #define FSCK_FAILED_INODE_BADFLUSH   −10013
476 #define FSCK_FAILED_INODE_FLUSH      −10014
477 #define FSCK_FAILED_BADREAD_INODE    −10015
478 #define FSCK_FAILED_BADREAD_INODE1   −10016
479 #define FSCK_FAILED_READ_INODE       −10017
480
481 #define FSCK_FAILED_IAG_BADFLUSH     −10018
482 #define FSCK_FAILED_IAG_FLUSH        −10019
483 #define FSCK_FAILED_BADREAD_IAG      −10020
484 #define FSCK_FAILED_BADREAD1_IAG     −10021
485 #define FSCK_FAILED_READ_IAG         −10022
486 #define FSCK_FAILED_IAG_CORRUPT_PXD  −10023
487
488 #define FSCK_FAILED_FBMAP_FLUSH      −10024
489 #define FSCK_FAILED_FBMAP_BADFLUSH   −10025
490 #define FSCK_FAILED_BADREAD_FBLKMP   −10026
491 #define FSCK_FAILED_READ_FBLKMP      −10027
492 #define FSCK_FAILED_WRITE_FBLKMP     −10028
493 #define FSCK_FAILED_BADWRITE_FBLKMP  −10029
494
495 #define FSCK_FAILED_PSBLK_WRITE      −10030
496 #define FSCK_FAILED_SSBLK_WRITE      −10031
497 #define FSCK_FAILED_BTHSBLK_WRITE    −10032
498 #define FSCK_FAILED_BTHSBLK_BAD      −10033
499
500 #define FSCK_FAILED_FSSIEXT_READ2    −10034
501 #define FSCK_FAILED_FSRTDIR_READ2    −10035
502 #define FSCK_FAILED_BADBLK_READ2     −10036
503 #define FSCK_FAILED_BMAP_READ2       −10037
504 #define FSCK_FAILED_LOG_READ2        −10038
505 #define FSCK_FAILED_SELF_READ2       −10039
506 #define FSCK_FAILED_SELF_READ3       −10040
507 #define FSCK_FAILED_SELF_READ4       −10041
508 #define FSCK_FAILED_SELF_READ5       −10042
509 #define FSCK_FAILED_SELF_NOWBAD      −10043
510 #define FSCK_FAILED_AGFS_READ2       −10044
511 #define FSCK_FAILED_AGFS_READ3       −10045
512 #define FSCK_FAILED_AGFS_READ4       −10046
513 #define FSCK_FAILED_AGFS_READ5       −10047
514 #define FSCK_FAILED_AGFS_NOWBAD      −10048
515
516 #define FSCK_FAILED_BOTHAITBAD       −10049
517 #define FSCK_FAILED_CANTREADAITEXT1  −10050
518 #define FSCK_FAILED_CANTREADAITEXT2  −10051
519 #define FSCK_FAILED_CANTREADAITEXT3  −10052
520 #define FSCK_FAILED_CANTREADAITEXT4  −10053
521 #define FSCK_FAILED_CANTREADAITEXT5  −10054
522 #define FSCK_FAILED_CANTREADAITEXT6  −10055
523 #define FSCK_FAILED_CANTREADAITEXT7  −10056
524 #define FSCK_FAILED_CANTREADAITEXT8  −10057
525 #define FSCK_FAILED_CANTREADAITEXT9  −10058
526 #define FSCK_FAILED_CANTREADAITEXTA  −10059
527 #define FSCK_FAILED_CANTREADAITEXTB  −10060
528 #define FSCK_FAILED_CANTREADAITEXTC  −10061
529 #define FSCK_FAILED_CANTREADAITEXTD  −10062
530 #define FSCK_FAILED_CANTREADAITEXTE  −10063
531 #define FSCK_FAILED_CANTREADAITEXTF  −10064
532 #define FSCK_FAILED_CANTREADAITEXTG  −10065
533 #define FSCK_FAILED_CANTREADAITEXTH  −10066
534 #define FSCK_FAILED_CANTREADAITEXTJ  −10067
535 #define FSCK_FAILED_CANTREADAITEXTK  −10068
536 #define FSCK_FAILED_CANTREADAITCTL   −10069
537 #define FSCK_FAILED_CANTREADAITS     −10070
538 #define FSCK_FAILED_CANTREADAIMNOW   −10071
539
540 #define FSCK_FAILED_IMPLF_BADFLUSH   −10072
```

```
#define FSCK_FAILED_IMPLF_FLUSH      −10073
#define FSCK_FAILED_BADREAD_IMPLF    −10074
#define FSCK_FAILED_READ_IMPLF       −10075

#define FSCK_FAILED_CANTREAD_DIRNOW  −10076
#define FSCK_FAILED_DIRGONEBAD       −10077
#define FSCK_FAILED_DIRGONEBAD2      −10078
#define FSCK_FAILED_DIRENTRYGONE     −10079
#define FSCK_FAILED_DIRENTRYBAD      −10080

#define FSCK_FAILED_MAPCTL_BADFLUSH  −10081
#define FSCK_FAILED_MAPCTL_FLUSH     −10082
#define FSCK_FAILED_BADREAD_MAPCTL   −10083
#define FSCK_FAILED_READ_MAPCTL      −10084

#define FSCK_FAILED_CANTREADBMPCTL   −10085

#define FSCK_FAILED_BMPLV_BADFLUSH   −10086
#define FSCK_FAILED_BMPLV_FLUSH      −10087
#define FSCK_FAILED_BADREAD_BMPLV    −10088
#define FSCK_FAILED_READ_BMPLV       −10089

#define FSCK_FAILED_BMPDM_BADFLUSH   −10090
#define FSCK_FAILED_BMPDM_FLUSH      −10091
#define FSCK_FAILED_BADREAD_BMPDM    −10092
#define FSCK_FAILED_READ_BMPDM       −10093
#define FSCK_FAILED_DYNSTG_EXHAUST1  −10094
#define FSCK_FAILED_DYNSTG_EXHAUST2  −10095
#define FSCK_FAILED_DYNSTG_EXHAUST3  −10096
#define FSCK_FAILED_DYNSTG_EXHAUST4  −10097
#define FSCK_FAILED_DYNSTG_EXHAUST5  −10098
#define FSCK_FAILED_DYNSTG_EXHAUST6  −10099
#define FSCK_FAILED_DYNSTG_EXHAUST7  −10100
#define FSCK_FAILED_DYNSTG_EXHAUST8  −10101
#define FSCK_FAILED_DYNSTG_EXHAUST9  −10102
#define FSCK_FAILED_DYNSTG_EXHAUSTA  −10103
#define FSCK_FAILED_REREAD_AGGINO    −10104

/* catastrophic error return codes */

#define FSCK_INTERNAL_ERROR_1   −11001
#define FSCK_INTERNAL_ERROR_2   −11002
#define FSCK_INTERNAL_ERROR_3   −11003
#define FSCK_INTERNAL_ERROR_4   −11004
#define FSCK_INTERNAL_ERROR_5   −11005
#define FSCK_INTERNAL_ERROR_6   −11006
#define FSCK_INTERNAL_ERROR_7   −11007
#define FSCK_INTERNAL_ERROR_8   −11008
#define FSCK_INTERNAL_ERROR_9   −11009
#define FSCK_INTERNAL_ERROR_10  −11010
#define FSCK_INTERNAL_ERROR_11  −11011
#define FSCK_INTERNAL_ERROR_12  −11012
#define FSCK_INTERNAL_ERROR_13  −11013
#define FSCK_INTERNAL_ERROR_14  −11014
#define FSCK_INTERNAL_ERROR_15  −11015
#define FSCK_INTERNAL_ERROR_16  −11016
#define FSCK_INTERNAL_ERROR_17  −11017
#define FSCK_INTERNAL_ERROR_18  −11018
#define FSCK_INTERNAL_ERROR_19  −11019
#define FSCK_INTERNAL_ERROR_20  −11020
#define FSCK_INTERNAL_ERROR_21  −11021
#define FSCK_INTERNAL_ERROR_22  −11022
#define FSCK_INTERNAL_ERROR_23  −11023
#define FSCK_INTERNAL_ERROR_24  −11024
#define FSCK_INTERNAL_ERROR_25  −11025
#define FSCK_INTERNAL_ERROR_26  −11026
#define FSCK_INTERNAL_ERROR_27  −11027
#define FSCK_INTERNAL_ERROR_28  −11028
#define FSCK_INTERNAL_ERROR_29  −11029
#define FSCK_INTERNAL_ERROR_30  −11030
#define FSCK_INTERNAL_ERROR_31  −11031
#define FSCK_INTERNAL_ERROR_32  −11032
#define FSCK_INTERNAL_ERROR_33  −11033
#define FSCK_INTERNAL_ERROR_34  −11034
#define FSCK_INTERNAL_ERROR_35  −11035
#define FSCK_INTERNAL_ERROR_36  −11036
#define FSCK_INTERNAL_ERROR_37  −11037
#define FSCK_INTERNAL_ERROR_38  −11038
#define FSCK_INTERNAL_ERROR_39  −11039
#define FSCK_INTERNAL_ERROR_40  −11040
#define FSCK_INTERNAL_ERROR_41  −11041
#define FSCK_INTERNAL_ERROR_42  −11042
#define FSCK_INTERNAL_ERROR_43  −11043
#define FSCK_INTERNAL_ERROR_44  −11044
#define FSCK_INTERNAL_ERROR_45  −11045
#define FSCK_INTERNAL_ERROR_46  −11046
#define FSCK_INTERNAL_ERROR_47  −11047
#define FSCK_INTERNAL_ERROR_48  −11048
#define FSCK_INTERNAL_ERROR_49  −11049
#define FSCK_INTERNAL_ERROR_50  −11050
```

```
631 #define FSCK_INTERNAL_ERROR_51  −11051
632 #define FSCK_INTERNAL_ERROR_52  −11052
633 #define FSCK_INTERNAL_ERROR_53  −11053
634 #define FSCK_INTERNAL_ERROR_54  −11054
635 #define FSCK_INTERNAL_ERROR_55  −11055
636 #define FSCK_INTERNAL_ERROR_56  −11056
637 #define FSCK_INTERNAL_ERROR_57  −11057
638 #define FSCK_INTERNAL_ERROR_58  −11058
639 #define FSCK_INTERNAL_ERROR_59  −11059
640 #define FSCK_INTERNAL_ERROR_60  −11060
641 #define FSCK_INTERNAL_ERROR_61  −11061
642 #define FSCK_INTERNAL_ERROR_62  −11062
643 #define FSCK_INTERNAL_ERROR_63  −11063
644 #define FSCK_INTERNAL_ERROR_64  −11064
645 #define FSCK_INTERNAL_ERROR_65  −11065
646 #define FSCK_INTERNAL_ERROR_66  −11066
647 #define FSCK_INTERNAL_ERROR_67  −11067
648 #define FSCK_INTERNAL_ERROR_68  −11068
649 #define FSCK_INTERNAL_ERROR_69  −11069
650 #define FSCK_INTERNAL_ERROR_70  −11070
651 #define FSCK_INTERNAL_ERROR_71  −11071
652 #define FSCK_INTERNAL_ERROR_72  −11072
653 #define FSCK_INTERNAL_ERROR_73  −11073
654 #define FSCK_INTERNAL_ERROR_74  −11074
655 #define FSCK_INTERNAL_ERROR_75  −11075
656 #define FSCK_INTERNAL_ERROR_76  −11076
657 #define FSCK_INTERNAL_ERROR_77  −11077
658 #define FSCK_INTERNAL_ERROR_78  −11078
659 #define FSCK_INTERNAL_ERROR_79  −11079
660 #define FSCK_INTERNAL_ERROR_80  −11080
661 #define FSCK_INTERNAL_ERROR_81  −11081
662 #define FSCK_INTERNAL_ERROR_82  −11082
663 #define FSCK_INTERNAL_ERROR_83  −11083
664 #define FSCK_INTERNAL_ERROR_84  −11084
665 #define FSCK_INTERNAL_ERROR_85  −11085
666 #define FSCK_INTERNAL_ERROR_86  −11086
667 #define FSCK_INTERNAL_ERROR_87  −11087
668 #define FSCK_INTERNAL_ERROR_88  −11088
669 #define FSCK_INTERNAL_ERROR_89  −11089
670
671 #endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_XFSCKINT
#define H_XFSCKINT

#ifndef _JFS_FSCKDIRE
#include "xfsck.h"
#endif

#include "xchkdsk.h"
#include "fsckmsgc.h"
#include "fsckpfs.h"
#include "fsckwsp.h"
#include "jfs_endian.h"

/*
  ---------------------- functions defined in fsckbmap ---------------------
*/

int rebuild_blkall_map(void);

int verify_blkall_map(void);

/*
  ---------------------- functions defined in fsckconn ---------------------
*/

int adjust_parents(struct fsck_inode_record *, uint32_t);

int check_connectedness(void);

int check_dir_integrity(void);

int check_link_counts(void);

/*
  ---------------------- functions defined in fsckdire ---------------------
*/

int fsck_dtDelete(struct dinode *, struct component_name *, uint32_t *);

int fsck_dtInsert(struct dinode *, struct component_name *, uint32_t *);

/*
  ---------------------- functions defined in fsckdtre ---------------------
*/

int direntry_add(struct dinode *, uint32_t, UniChar *);

int direntry_get_inonum(uint32_t, int, UniChar *, int, UniChar *, uint32_t *);

int direntry_get_objnam(uint32_t, uint32_t, int *, UniChar *);

int direntry_remove(struct dinode *, uint32_t);

int dTree_processing(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *, int);

int dTree_search(struct dinode *, UniChar *, uint32_t, UniChar *,
                 uint32_t, struct dtslot **, int8_t *, struct fsck_inode_record *);

int find_first_dir_leaf(struct dinode *, dtpage_t **, int64_t *, int8_t *, int8_t *);

void init_dir_tree(dtroot_t *);

int process_valid_dir_data(struct dinode *, uint32_t, struct fsck_inode_record *,
                           struct fsck_ino_msg_info *, int);

int reconnect_fs_inodes(void);

/*
  ---------------------- functions defined in fsckimap ---------------------
*/

int AIS_redundancy_check(void);

```

```
91  int AIS_replication(void);
92
93  int rebuild_agg_iamap(void);
94
95  int rebuild_fs_iamaps(void);
96
97  int record_dupchk_inode_extents(void);
98
99  int verify_agg_iamap(void);
100
101 int verify_fs_iamaps(void);
102
103 /*
104   ---------------------- functions defined in fsckino ---------------------
105 */
106
107 #define inode_type_recognized(iptr)\
108                ( ISDIR(iptr->di_mode) || \
109                  ISREG(iptr->di_mode) || \
110                  ISLNK(iptr->di_mode) || \
111                  ISBLK(iptr->di_mode) || \
112                  ISCHR(iptr->di_mode) || \
113                  ISFIFO(iptr->di_mode)|| \
114                  ISSOCK(iptr->di_mode)       )
115
116 int backout_ACL(struct dinode *, struct fsck_inode_record *);
117
118 int backout_EA(struct dinode *, struct fsck_inode_record *);
119
120 int calculate_dasd_used(void);
121
122 int clear_ACL_field(struct fsck_inode_record *, struct dinode *);
123
124 int clear_EA_field(struct fsck_inode_record *, struct dinode *);
125
126 int display_path(uint32_t, int, uint32_t, char *, struct fsck_inode_record *);
127
128 int display_paths(uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
129
130 int first_ref_check_inode(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
131
132 int get_path(uint32_t, uint32_t, char **, struct fsck_inode_record *);
133
134 int in_inode_data_check(struct fsck_inode_record *, struct fsck_ino_msg_info *);
135
136 int inode_is_in_use(struct dinode *, uint32_t);
137
138 int parent_count(struct fsck_inode_record *);
139
140 int record_valid_inode(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
141
142 int release_inode(uint32_t, struct fsck_inode_record *, struct dinode *);
143
144 int unrecord_valid_inode(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
145
146 int validate_ACL(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
147
148 int validate_dasd_used(void);
149
150 int validate_data(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
151
152 int validate_dir_data(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
153
154 int validate_EA(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *);
155
156 int validate_record_fileset_inode(uint32_t, uint32_t, struct dinode *, struct fsck_ino_msg_info *);
157
158 /*
159   ---------------------- functions defined in fsckmeta --------------------
160 */
161
162 int agg_clean_or_dirty(void);
163
164 int fatal_dup_check(void);
165
166 int first_ref_check_agg_metadata(void);
167
168 int first_ref_check_fixed_metadata(void);
169
170 int first_ref_check_fs_metadata(void);
171
172 int first_ref_check_inode_extents(void);
173
174 int record_fixed_metadata(void);
175
176 int replicate_superblock(void);
177
178 int validate_fs_metadata(void);
179
180 int validate_repair_superblock(void);
```

```
181
182 int validate_select_agg_inode_table(void);
183
184 /*
185   ---------------------- functions defined in fsckmsgs ---------------------
186 */
187
188 void fsck_send_msg(int, int);
189
190 /*
191   ---------------------- functions defined in fsckpfs ---------------------
192 */
193
194 int ait_node_get(int64_t, xtpage_t *);
195
196 int ait_node_put(int64_t, xtpage_t *);
197
198 int ait_special_read_ext1(int);
199
200 int blkmap_find_bit(int64_t, int64_t *, uint32_t *, uint32_t *);
201
202 int blkmap_flush(void);
203
204 int blkmap_get_ctl_page(struct fsck_blk_map_hdr *);
205
206 int blkmap_get_page(int64_t, struct fsck_blk_map_page **);
207
208 int blkmap_put_ctl_page(struct fsck_blk_map_hdr *);
209
210 int blkmap_put_page(int64_t);
211
212 int blktbl_ctl_page_put(struct dbmap *);
213
214 int blktbl_dmap_get(int64_t, struct dmap **);
215
216 int blktbl_dmap_put(struct dmap *);
217
218 int blktbl_dmaps_flush(void);
219
220 int blktbl_Ln_page_get(int8_t, int64_t, struct dmapctl **);
221
222 int blktbl_Ln_page_put(struct dmapctl *);
223
224 int blktbl_Ln_pages_flush(void);
225
226 int close_volume(void);
227
228 int default_volume(void);
229
230 int dnode_get(int64_t, uint32_t, dtpage_t **);
231
232 int ea_get(int64_t, uint32_t, char *, uint32_t *, uint32_t *, int64_t *);
233
234 int fscklog_put_buffer(void);
235
236 int iag_get(int, int, int, int32_t, struct iag **);
237
238 int iag_get_first(int, int, int, struct iag **);
239
240 int iag_get_next(struct iag **);
241
242 int iag_put(struct iag *);
243
244 int iags_flush(void);
245
246 int inodes_flush(void);
247
248 int inode_get(int, int, uint32_t, struct dinode **);
249
250 int inode_get_first_fs(int, uint32_t *, struct dinode **);
251
252 int inode_get_next(uint32_t *, struct dinode **);
253
254 int inode_put(struct dinode *);
255
256 int inotbl_get_ctl_page(int, struct dinomap **);
257
258 int inotbl_put_ctl_page(int, struct dinomap *);
259
260 int mapctl_get(int64_t, void **);
261
262 int mapctl_put(void *);
263
264 int mapctl_flush(void);
265
266 int node_get(int64_t, xtpage_t **);
267
268 int open_volume(char *);
269
270 int readwrite_device(int64_t, uint32_t, uint32_t *, void *, int);
```

```
271
272 int recon_dnode_assign(int64_t, dtpage_t **);
273
274 int recon_dnode_get(int64_t, dtpage_t **);
275
276 int recon_dnode_put(dtpage_t *);
277
278 int recon_dnode_release(dtpage_t *);
279
280 /*
281   ---------------------- functions defined in fsckruns ----------------------
282 */
283
284 void fsck_hbeat_start(void);
285
286 void fsck_hbeat_stop(void);
287
288 /*
289   ---------------------- functions defined in fsckwsp ----------------------
290 */
291
292 int alloc_vlarge_buffer(void);
293
294 int alloc_wrksp(uint32_t, int, int, void **);   /* called from both fsck modules
295                                                  * and from logredo modules
296                                                  */
297
298 int blkall_decrement_owners(int64_t);
299
300 int blkall_increment_owners(int64_t);
301
302 int blkall_ref_check(int64_t, int *);
303
304 int dire_buffer_alloc(dtpage_t **);
305
306 int dire_buffer_release(dtpage_t *);
307
308 int directory_buffers_alloc(void);
309
310 int directory_buffers_release(void);
311
312 int dtreeQ_dequeue(struct dtreeQelem **);
313
314 int dtreeQ_enqueue(struct dtreeQelem *);
315
316 int dtreeQ_get_elem(struct dtreeQelem **);
317
318 int dtreeQ_rel_elem(struct dtreeQelem *);
319
320 int establish_agg_workspace(void);
321
322 int establish_ea_iobuf(void);
323
324 int establish_fs_workspace(void);
325
326 int establish_io_buffers(void);
327
328 int establish_wsp_block_map_ctl(void);
329
330 int extent_record(int64_t, int64_t);
331
332 int extent_unrecord(int64_t, int64_t);
333
334 int fsck_alloc_fsblks(int32_t, int64_t *);
335
336 int fsck_dealloc_fsblks(int32_t, int64_t);
337
338 int fscklog_end(void);
339
340 int fscklog_init(void);
341
342 int fscklog_start(void);
343
344 int get_inode_extension(struct fsck_inode_ext_record **);
345
346 int get_inorecptr(int, int, uint32_t, struct fsck_inode_record **);
347
348 int get_inorecptr_first(int, uint32_t *, struct fsck_inode_record **);
349
350 int get_inorecptr_next(int, uint32_t *, struct fsck_inode_record **);
351
352 int init_agg_record(void);
353
354 int process_extent(struct fsck_inode_record *, uint32_t, int64_t, int8_t,
355                    int8_t, struct fsck_ino_msg_info *, uint32_t *, int8_t *, int);
356
357 int release_inode_extension(struct fsck_inode_ext_record *);
358
359 int release_logredo_allocs(void);
360
```

```
361 int temp_inode_buf_alloc(char **);
362
363 int temp_inode_buf_release(char *);
364
365 int temp_node_buf_alloc(char **);
366
367 int temp_node_buf_release(char *);
368
369 int treeQ_dequeue(struct treeQelem **);
370
371 int treeQ_enqueue(struct treeQelem *);
372
373 int treeQ_get_elem(struct treeQelem **);
374
375 int treeQ_rel_elem(struct treeQelem *);
376
377 int treeStack_pop(struct fsck_inode_record **);
378
379 int treeStack_push(struct fsck_inode_record *);
380
381 int workspace_release(void);
382
383 /*
384   ---------------------- functions defined in fsckxtre ---------------------
385 */
386
387 int find_first_leaf(struct dinode *, xtpage_t **, int64_t *, int8_t *, int8_t *);
388
389 int init_xtree_root(struct dinode *);
390
391 int process_valid_data(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *, int);
392
393 int xTree_processing(struct dinode *, uint32_t, struct fsck_inode_record *, struct fsck_ino_msg_info *, int);
394
395 int xTree_search(struct dinode *, int64_t, xad_t **, int8_t *);
396
397 /*
398   ---------------------- functions defined in xchkdsk.c ---------------------
399 */
400
401 void report_readait_error(int, int, int);
402
403 #endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_BTREE
#define _H_JFS_BTREE
/*
 *      jfs_btree.h: B+−tree
 *
 * JFS B+−tree (dtree and xtree) common definitions
 */

/*
 *      basic btree page − btpage
 */
struct btpage {
        int64_t next;            /* 8: right sibling bn */
        int64_t prev;            /* 8: left sibling bn */

        uint8_t flag;           /* 1: */
        uint8_t rsrvd[7];       /* 7: type specific */
        int64_t self;           /* 8: self address */

        uint8_t entry[4064];    /* 4064: */
};                              /* (4096) */

/* btpage flag */
#define BT_TYPE       0x07      /* B+−tree index */
#define BT_ROOT       0x01      /* root page */
#define BT_LEAF       0x02      /* leaf page */
#define BT_INTERNAL   0x04      /* internal page */
#define BT_RIGHTMOST  0x10      /* rightmost page */
#define BT_LEFTMOST   0x20      /* leftmost page */
#define BT_SWAPPED    0x80      /* endian swapped when read from disk */

#endif                          /* _H_JFS_BTREE */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_DINODE
#define _H_JFS_DINODE

/*
 *      jfs_dinode.h: on−disk inode manager
 *
 */
#include "jfs_types.h"
#include "jfs_dtree.h"
#include "jfs_xtree.h"

#define INODESLOTSIZE           128
#define L2INODESLOTSIZE         7
#define log2INODESIZE           9       /* log2(bytes per dinode) */


/*
 *      on−disk inode (struct dinode): 512 bytes
 *
 * note: align 64−bit fields on 8−byte boundary.
 */
struct dinode {
        /*
         *      I. base area (128 bytes)
         *      −−−−−−−−−−−−−−−−−−−−−−−−
         *
         * define generic/POSIX attributes
         */
        uint32_t  di_inostamp;  /* 4: stamp to show inode belongs to fileset */
        int32_t   di_fileset;   /* 4: fileset number */
        uint32_t  di_number;    /* 4: inode number, aka file serial number */
        uint32_t  di_gen;               /* 4: inode generation number */

        pxd_t     di_ixpxd;             /* 8: inode extent descriptor */

        int64_t   di_size;              /* 8: size */
        int64_t   di_nblocks;   /* 8: number of blocks allocated */

        uint32_t  di_nlink;             /* 4: number of links to the object */

        uint32_t  di_uid;               /* 4: user id of owner */
        uint32_t  di_gid;               /* 4: group id of owner */

        uint32_t  di_mode;              /* 4: attribute, format and permission */

        struct timestruc_t di_atime;    /* 8: time last data accessed */
        struct timestruc_t di_ctime;    /* 8: time last status changed */
        struct timestruc_t di_mtime;    /* 8: time last data modified */
        struct timestruc_t di_otime;    /* 8: time created */

        dxd_t     di_acl;               /* 16: acl descriptor */

        dxd_t     di_ea;                /* 16: ea descriptor */

        int32_t   di_next_index;/* 4: Next available dir_table index */

        int32_t   di_acltype;   /* 4: Type of ACL */

        /*
         *      Extension Areas.
         *
         *      Historically, the inode was partitioned into 4 128−byte areas,
         *      the last 3 being defined as unions which could have multiple
         *      uses.  The first 96 bytes had been completely unused until
         *      an index table was added to the directory.  It is now more
         *      useful to describe the last 3/4 of the inode as a single
         *      union.  We would probably be better off redesigning the
         *      entire structure from scratch, but we don't want to break
         *      commonality with OS/2's JFS at this time.
         */
        union {
                struct {
                        /*
```

```
                         * This table contains the information needed to
                         * find a directory entry from a 32-bit index.
                         * If the index is small enough, the table is inline,
                         * otherwise, an x-tree root overlays this table
                         */
                        struct dir_table_slot _table[12];       /* 96: inline */

                        dtroot_t         _dtroot;               /* 288: dtree root */
                } _dir;                                             /* (384) */
#define di_dirtable     u._dir._table
#define di_dtroot       u._dir._dtroot
#define di_parent   di_dtroot.header.idotdot
#define di_DASD         di_dtroot.header.DASD

                struct {
                        union {
                                uint8_t  _data[96];             /* 96: unused */
                                struct {
                                        void      *_imap;       /* 4: unused */
                                        uint32_t  _gengen;      /* 4: generator */
                                } _imap;
                        } _u1;                                  /* 96: */
#define di_gengen       u._file._u1._imap._gengen

                        union {
                                uint8_t  _xtroot[288];
                                struct {
                                        uint8_t  unused[16];    /* 16: */
                                        dxd_t    _dxd;          /* 16: */
                                        union {
                                                uint32_t  _rdev;        /* 4: */
                                                uint8_t   _fastsymlink[128];
                                        } _u;
                                        uint8_t  _inlineea[128];
                                } _special;
                        } _u2;
                } _file;
#define di_xtroot           u._file._u2._xtroot
#define di_dxd              u._file._u2._special._dxd
#define di_btroot           di_xtroot
#define di_inlinedata   u._file._u2._special._u
#define di_rdev             u._file._u2._special._u._rdev
#define di_fastsymlink  u._file._u2._special._u._fastsymlink
#define di_inlineea     u._file._u2._special._inlineea
        } u;
};

/* di_mode */
/*
 * The utilities that are dealing directly with the disk
 * i-node define the modes as follows. The filesystem itself
 * should use the standard S_IFMT, etc. defines in stat.h
 */
#define IFMT    0xF000          /* S_IFMT - mask of file type */
#define IFDIR   0x4000          /* S_IFDIR - directory */
#define IFREG   0x8000          /* S_IFREG - regular file */
#define IFLNK   0xA000          /* S_IFLNK - symbolic link */
#define IFBLK   0x6000          /* S_IFBLK - block special file */
#define IFCHR   0x2000          /* S_IFCHR - character special file */
#define IFFIFO  0x1000          /* S_IFFIFO - FIFO */
#define IFSOCK  0xC000          /* S_IFSOCK - socket */

#define ISUID   0x0800          /* S_ISUID - set user id when exec'ing */
#define ISGID   0x0400          /* S_ISGID - set group id when exec'ing */

#define IREAD   0x0100          /* S_IRUSR - read permission */
#define IWRITE  0x0080          /* S_IWUSR - write permission */
#define IEXEC   0x0040          /* S_IXUSR - execute permission */

/* extended mode bits (on-disk inode di_mode) */
#define IFJOURNAL   0x00010000  /* journalled file */
#define ISPARSE     0x00020000  /* sparse file enabled */
#define INLINEEA    0x00040000  /* inline EA area free */
#define ISWAPFILE       0x00800000      /* file open for pager swap space */

/* more extended mode bits */
#define IDIRECTORY      0x20000000      /* directory (shadow of real bit) */

#endif /*_H_JFS_DINODE */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_DMAP
#define _H_JFS_DMAP

#include "jfs_types.h"

#define BMAPVERSION     1       /* version number */
#define TREESIZE        (256+64+16+4+1) /* size of a dmap tree */
#define LEAFIND         (64+16+4+1)     /* index of 1st leaf of a dmap tree */
#define LPERDMAP        256     /* num leaves per dmap tree */
#define L2LPERDMAP      8       /* l2 number of leaves per dmap tree */
#define DBWORD          32      /* # of blks covered by a map word */
#define L2DBWORD        5       /* l2 # of blks covered by a mword */
#define BUDMIN          L2DBWORD        /* max free string in a map word */
#define BPERDMAP        (LPERDMAP * DBWORD)     /* num of blks per dmap */
#define L2BPERDMAP      13      /* l2 num of blks per dmap */
#define CTLTREESIZE     (1024+256+64+16+4+1)    /* size of a dmapctl tree */
#define CTLLEAFIND      (256+64+16+4+1) /* idx of 1st leaf of a dmapctl tree */
#define LPERCTL         1024    /* num of leaves per dmapctl tree */
#define L2LPERCTL       10      /* l2 num of leaves per dmapctl tree */
#define ROOT            0       /* index of the root of a tree */
#define NOFREE          ((int8_t) −1)   /* no blocks free */
#define MAXAG           128     /* max number of allocation groups */
#define L2MAXAG         7       /* l2 max num of AG */
#define L2MINAGSZ       25      /* l2 of minimum AG size in bytes */
#define BMAPBLKNO       0       /* lblkno of bmap within the map */

/*
 * maximum l2 number of disk blocks at the various dmapctl levels.
 */
#define L2MAXL0SIZE     (L2BPERDMAP + 1 * L2LPERCTL)
#define L2MAXL1SIZE     (L2BPERDMAP + 2 * L2LPERCTL)
#define L2MAXL2SIZE     (L2BPERDMAP + 3 * L2LPERCTL)

/*
 * maximum number of disk blocks at the various dmapctl levels.
 */
#define MAXL0SIZE       ((int64_t)1 << L2MAXL0SIZE)
#define MAXL1SIZE       ((int64_t)1 << L2MAXL1SIZE)
#define MAXL2SIZE       ((int64_t)1 << L2MAXL2SIZE)

#define MAXMAPSIZE      MAXL2SIZE       /* maximum aggregate map size */

/*
 * determine the maximum free string for four (lower level) nodes
 * of the tree.
 */
#define TREEMAX(cp)                                             \
        ((signed char)(MAX(MAX(*(cp),*((cp)+1)),        \
                           MAX(*((cp)+2),*((cp)+3)))))

/*
 * convert disk block number to the logical block number of the dmap
 * describing the disk block.  s is the log2(number of logical blocks per page)
 *
 * The calculation figures out how many logical pages are in front of the dmap.
 *      − the number of dmaps preceding it
 *      − the number of L0 pages preceding its L0 page
 *      − the number of L1 pages preceding its L1 page
 *      − 3 is added to account for the L2, L1, and L0 page for this dmap
 *      − 1 is added to account for the control page of the map.
 */
#define BLKTODMAP(b,s)    \
        ((((b) >> 13) + ((b) >> 23) + ((b) >> 33) + 3 + 1) << (s))

/*
 * convert disk block number to the logical block number of the LEVEL 0
 * dmapctl describing the disk block.  s is the log2(number of logical blocks
 * per page)
 *
 * The calculation figures out how many logical pages are in front of the L0.
 *      − the number of dmap pages preceding it
 *      − the number of L0 pages preceding it
 *      − the number of L1 pages preceding its L1 page
```

```
 *      − 2 is added to account for the L2, and L1 page for this L0
 *      − 1 is added to account for the control page of the map.
 */
#define BLKTOL0(b,s)      \
        (((((b) >> 23) << 10) + ((b) >> 23) + ((b) >> 33) + 2 + 1) << (s))

/*
 * convert disk block number to the logical block number of the LEVEL 1
 * dmapctl describing the disk block.  s is the log2(number of logical blocks
 * per page)
 *
 * The calculation figures out how many logical pages are in front of the L1.
 *      − the number of dmap pages preceding it
 *      − the number of L0 pages preceding it
 *      − the number of L1 pages preceding it
 *      − 1 is added to account for the L2 page
 *      − 1 is added to account for the control page of the map.
 */
#define BLKTOL1(b,s)      \
     (((((b) >> 33) << 20) + (((b) >> 33) << 10) + ((b) >> 33) + 1 + 1) << (s))

/*
 * convert disk block number to the logical block number of the dmapctl
 * at the specified level which describes the disk block.
 */
#define BLKTOCTL(b,s,l)   \
        ((((l) == 2) ? 1 : ((l) == 1) ? BLKTOL1((b),(s)) : BLKTOL0((b),(s)))

/*
 * convert aggregate map size to the zero origin dmapctl level of the
 * top dmapctl.
 */
#define BMAPSZTOLEV(size)       \
        (((size) <= MAXL0SIZE) ? 0 : ((size) <= MAXL1SIZE) ? 1 : 2)

/* convert disk block number to allocation group number.
 */
#define BLKTOAG(b,sb)   ((b) >> ((sb)->s_jfs_bmap->db_agl2size))

/* convert allocation group number to starting disk block
 * number.
 */
#define AGTOBLK(a,ip)   \
        ((int64_t)(a) << ((ip)->i_sb->s_jfs_bmap->db_agl2size))

/*
 *      dmap summary tree
 *
 * struct dmaptree must be consistent with struct dmapctl.
 */
struct dmaptree {
        int32_t  nleafs;                /* 4: number of tree leafs      */
        int32_t  l2nleafs;              /* 4: l2 number of tree leafs   */
        int32_t  leafidx;               /* 4: index of first tree leaf  */
        int32_t  height;                /* 4: height of the tree        */
        int8_t   budmin;                /* 1: min l2 tree leaf value to combine */
        int8_t   stree[TREESIZE];       /* TREESIZE: tree               */
        uint8_t  pad[2];                /* 2: pad to word boundary      */
};                          /* − 360 −                      */

/*
 *      dmap page per 8K blocks bitmap
 */
struct dmap {
        int32_t     nblocks;            /* 4: num blks covered by this dmap     */
        int32_t     nfree;                  /* 4: num of free blks in this dmap     */
        int64_t     start;                  /* 8: starting blkno for this dmap      */
        struct dmaptree tree;           /* 360: dmap tree                       */
        uint8_t     pad[1672];          /* 1672: pad to 2048 bytes              */
        uint32_t    wmap[LPERDMAP];     /* 1024: bits of the working map        */
        uint32_t    pmap[LPERDMAP];     /* 1024: bits of the persistent map     */
};                                          /* − 4096 −                             */

/*
 *      disk map control page per level.
 *
 * struct dmapctl must be consistent with struct dmaptree.
 */
struct dmapctl {
        int32_t  nleafs;                /* 4: number of tree leafs         */
        int32_t  l2nleafs;              /* 4: l2 number of tree leafs      */
        int32_t  leafidx;               /* 4: index of the first tree leaf */
        int32_t  height;                /* 4: height of tree               */
        int8_t   budmin;                /* 1: minimum l2 tree leaf value   */
        int8_t   stree[CTLTREESIZE];/* CTLTREESIZE: dmapctl tree   */
        uint8_t  pad[2714];             /* 2714: pad to 4096               */
};                              /* − 4096 −                        */

/*
 *      common definition for dmaptree within dmap and dmapctl
```

```
 */
typedef union {
        struct dmaptree t1;
        struct dmapctl t2;
} dmtree_t;

/* macros for accessing fields within dmtree_t */
#define dmt_nleafs        t1.nleafs
#define dmt_l2nleafs  t1.l2nleafs
#define dmt_leafidx   t1.leafidx
#define dmt_height        t1.height
#define dmt_budmin        t1.budmin
#define dmt_stree         t1.stree

/*
 *      on−disk aggregate disk allocation map descriptor.
 */
struct dbmap {
        int64_t  dn_mapsize;            /* 8: number of blocks in aggregate     */
        int64_t  dn_nfree;                  /* 8: num free blks in aggregate map    */
        int32_t  dn_l2nbperpage;        /* 4: number of blks per page           */
        int32_t  dn_numag;                  /* 4: total number of ags               */
        int32_t  dn_maxlevel;       /* 4: number of active ags              */
        int32_t  dn_maxag;                  /* 4: max active alloc group number     */
        int32_t  dn_agpref;             /* 4: preferred alloc group (hint)      */
        int32_t  dn_aglevel;            /* 4: dmapctl level holding the AG      */
        int32_t  dn_agheigth;       /* 4: height in dmapctl of the AG       */
        int32_t  dn_agwidth;            /* 4: width in dmapctl of the AG        */
        int32_t  dn_agstart;            /* 4: start tree index at AG height     */
        int32_t  dn_agl2size;       /* 4: l2 num of blks per alloc group    */
        int64_t  dn_agfree[MAXAG];  /* 8*MAXAG: per AG free count           */
        int64_t  dn_agsize;                 /* 8: num of blks per alloc group       */
        int8_t   dn_maxfreebud;     /* 1: max free buddy system             */
        uint8_t  pad[3007];                 /* 3007: pad to 4096                    */
};                                  /* − 4096 −                             */

#endif                          /* _H_JFS_DMAP */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_DTREE
#define _H_JFS_DTREE

/*
 *      jfs_dtree.h: directory B+−tree manager
 */
#include "jfs_btree.h"

typedef union {
        struct {
                int             tid;
                struct inode  *ip;
                uint32_t        ino;
        } leaf;
        pxd_t xd;
} ddata_t;


/*
 *      entry segment/slot
 *
 * an entry consists of type dependent head/only segment/slot and
 * additional segments/slots linked vi next field;
 * N.B. last/only segment of entry is terminated by next = −1;
 */
/*
 *      directory page slot
 */
struct dtslot {
        int8_t   next;          /*  1: */
        int8_t   cnt;           /*  1: */
        UniChar  name[15];      /* 30: */
};                              /* (32) */


#define DATASLOTSIZE    16
#define L2DATASLOTSIZE   4
#define DTSLOTSIZE      32
#define L2DTSLOTSIZE     5
#define DTSLOTHDRSIZE    2
#define DTSLOTDATASIZE  30
#define DTSLOTDATALEN   15

/*
 *       internal node entry head/only segment
 */
struct idtentry {
        pxd_t    xd;            /* 8: child extent descriptor */

        int8_t   next;          /* 1: */
        uint8_t  namlen;        /* 1: */
        UniChar  name[11];      /* 22: 2−byte aligned */
};                              /* (32) */

#define DTIHDRSIZE      10
#define DTIHDRDATALEN   11

/* compute number of slots for entry */
#define NDTINTERNAL(klen) ( ((4 + (klen)) + (15 − 1)) / 15 )


/*
 *      leaf node entry head/only segment
 *
 *      For legacy filesystems, name contains 13 unichars −− no index field
 */
struct ldtentry {
        uint32_t  inumber;      /* 4: 4−byte aligned */
        int8_t    next;         /* 1: */
        uint8_t   namlen;       /* 1: */
        UniChar   name[11];     /* 22: 2−byte aligned */
        uint32_t  index;        /* 4: index into dir_table */
};                              /* (32) */
```

```
91
92  #define DTLHDRSIZE              6
93  #define DTLHDRDATALEN_LEGACY    13      /* Old (OS/2) format */
94  #define DTLHDRDATALEN           11
95
96  /*
97   * dir_table used for directory traversal during readdir
98   */
99
100 /*
101  * Keep persistent index for directory entries
102  */
103 #define DO_INDEX(INODE) ((INODE)->i_sb->s_jfs_mntflag & JFS_DIR_INDEX)
104
105 /*
106  * Maximum entry in inline directory table
107  */
108 #define MAX_INLINE_DIRTABLE_ENTRY 13
109
110 struct dir_table_slot {
111         uint8_t   rsrvd;  /* 1: */
112         uint8_t   flag;   /* 1: 0 if free */
113         uint8_t   slot;   /* 1: slot within leaf page of entry */
114         uint8_t   addr1;  /* 1: upper 8 bits of leaf page address */
115         uint32_t  addr2;  /* 4: lower 32 bits of leaf page address −OR− index
116                              of next entry when this entry was deleted */
117 };                        /* (8) */
118
119 /*
120  * flag values
121  */
122 #define DIR_INDEX_VALID 1
123 #define DIR_INDEX_FREE  0
124
125 /* compute number of slots for entry */
126 #define NDTLEAF_LEGACY(klen)    ( ((2 + (klen)) + (15 − 1)) / 15 )
127 #define NDTLEAF NDTINTERNAL
128
129
130 /*
131  *      directory root page (in−line in on−disk inode):
132  *
133  * cf. dtpage_t below.
134  */
135 typedef union {
136         struct {
137                 struct dasd DASD;           /* 16: DASD limit/usage info  F226941 */
138
139                 uint8_t   flag;     /* 1: */
140                 int8_t    nextindex;/* 1: next free entry in stbl */
141                 int8_t    freecnt;      /* 1: free count */
142                 int8_t    freelist;     /* 1: freelist header */
143
144                 uint32_t  idotdot;      /* 4: parent inode number */
145
146                 int8_t    stbl[8];      /* 8: sorted entry index table */
147         } header;                       /* (32) */
148
149         struct dtslot  slot[9];
150 } dtroot_t;
151
152 #define DTROOTMAXSLOT   9
153
154 /*
155  *      directory regular page:
156  *
157  *      entry slot array of 32 byte slot
158  *
159  * sorted entry slot index table (stbl):
160  * contiguous slots at slot specified by stblindex,
161  * 1−byte per entry
162  *   512 byte block:  16 entry tbl (1 slot)
163  *  1024 byte block:  32 entry tbl (1 slot)
164  *  2048 byte block:  64 entry tbl (2 slot)
165  *  4096 byte block: 128 entry tbl (4 slot)
166  *
167  * data area:
168  *   512 byte block:  16 − 2 =  14 slot
169  *  1024 byte block:  32 − 2 =  30 slot
170  *  2048 byte block:  64 − 3 =  61 slot
171  *  4096 byte block: 128 − 5 = 123 slot
172  *
173  * N.B. index is 0−based; index fields refer to slot index
174  * except nextindex which refers to entry index in stbl;
175  * end of entry stot list or freelist is marked with −1.
176  */
177 typedef union {
178         struct {
179                 int64_t  next;      /* 8: next sibling */
180                 int64_t  prev;      /* 8: previous sibling */
```

```
181
182                 uint8_t  flag;      /* 1: */
183                 int8_t   nextindex;     /* 1: next entry index in stbl */
184                 int8_t   freecnt;       /* 1: */
185                 int8_t   freelist;      /* 1: slot index of head of freelist */
186
187                 uint8_t  maxslot;       /* 1: number of slots in page slot[] */
188                 int8_t   stblindex;     /* 1: slot index of start of stbl */
189                 uint8_t  rsrvd[2];      /* 2: */
190
191                 pxd_t    self;      /* 8: self pxd */
192         } header;                       /* (32) */
193
194         struct dtslot  slot[128];
195 } dtpage_t;
196
197 #define DTPAGEMAXSLOT        128
198
199 #define DT8THPGNODEBYTES     512
200 #define DT8THPGNODETSLOTS      1
201 #define DT8THPGNODESLOTS      16
202
203 #define DTQTRPGNODEBYTES    1024
204 #define DTQTRPGNODETSLOTS      1
205 #define DTQTRPGNODESLOTS      32
206
207 #define DTHALFPGNODEBYTES   2048
208 #define DTHALFPGNODETSLOTS     2
209 #define DTHALFPGNODESLOTS     64
210
211 #define DTFULLPGNODEBYTES   4096
212 #define DTFULLPGNODETSLOTS     4
213 #define DTFULLPGNODESLOTS    128
214
215 #define DTENTRYSTART    1
216
217 /* get sorted entry table of the page */
218 #define DT_GETSTBL(p) ( ((p)->header.flag & BT_ROOT) ?\
219         ((dtroot_t *)(p))->header.stbl : \
220         (int8_t *)&(p)->slot[(p)->header.stblindex] )
221
222 /*
223  * Flags for dtSearch
224  */
225 #define JFS_CREATE 1
226 #define JFS_LOOKUP 2
227 #define JFS_REMOVE 3
228 #define JFS_RENAME 4
229
230 #endif                          /* !_H_JFS_DTREE */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_FILSYS
#define _H_JFS_FILSYS

/*
 *      jfs_filsys.h
 *
 * file system (implementation−dependent) constants
 *
 * refer to <limits.h> for system wide implementation−dependent constants
 */

#include "jfs_types.h"

/*
 *       file system option (superblock flag)
 */
/* platform option (conditional compilation) */
#define JFS_AIX             0x80000000  /* AIX support */
/*      POSIX name/directory  support */

#define JFS_OS2             0x40000000  /* OS/2 support */
/*      case−insensitive name/directory support */

#define JFS_DFS             0x20000000  /* DCE DFS LFS support */

#define JFS_LINUX       0x10000000      /* Linux support */
/*      case−sensitive name/directory support */

/* directory option */
#define JFS_UNICODE         0x00000001  /* unicode name */

/* commit option */
#define JFS_COMMIT          0x00000f00  /* commit option mask */
#define JFS_GROUPCOMMIT 0x00000100      /* group (of 1) commit */
#define JFS_LAZYCOMMIT  0x00000200      /* lazy commit */
#define JFS_TMPFS           0x00000400  /* temporary file system −
                                         * do not log/commit:
                                         */

/* log logical volume option */
#define JFS_INLINELOG   0x00000800      /* inline log within file system */
#define JFS_INLINEMOVE  0x00001000      /* inline log being moved */

/* Secondary aggregate inode table */
#define JFS_BAD_SAIT    0x00010000      /* current secondary ait is bad */

/* sparse regular file support */
#define JFS_SPARSE          0x00020000  /* sparse regular file */

/* DASD Limits          F226941 */
#define JFS_DASD_ENABLED        0x00040000      /* DASD limits enabled */
#define JFS_DASD_PRIME          0x00080000      /* Prime DASD usage on boot */

/* Directory index */
#define JFS_DIR_INDEX   0x00200000      /* Persistant index for */
                                                /* directory entries    */


/*
 *      buffer cache configuration
 */
/* page size */
#ifdef PSIZE
#undef PSIZE
#endif
#define PSIZE           4096    /* page size (in byte) */
#define L2PSIZE           12    /* log2(PSIZE) */
#define POFFSET         4095    /* offset within page */

/* buffer page size */
#define BPSIZE  PSIZE

/*
```

```
 *      fs fundamental size
 *
 * PSIZE >= file system block size >= PBSIZE >= DISIZE
 */
#define PBSIZE          512     /* physical block size (in byte) */
#define L2PBSIZE          9     /* log2(PBSIZE) */

#define DISIZE          512     /* on-disk inode size (in byte) */
#define L2DISIZE          9     /* log2(DISIZE) */

#define IDATASIZE       256     /* inode inline data size */
#define IXATTRSIZE      128     /* inode inline extended attribute size */

#define XTPAGE_SIZE     4096
#define log2_PAGESIZE     12

#define IAG_SIZE        4096
#define IAG_EXTENT_SIZE 4096
#define INOSPERIAG          4096        /* number of disk inodes per iag */
#define L2INOSPERIAG      12    /* l2 number of disk inodes per iag */
#define INOSPEREXT           32         /* number of disk inode per extent */
#define L2INOSPEREXT       5    /* l2 number of disk inode per extent */
#define IXSIZE          (DISIZE * INOSPEREXT)   /* inode extent size */
#define INOSPERPAGE            8        /* number of disk inodes per 4K page */
#define L2INOSPERPAGE      3    /* log2(INOSPERPAGE) */

#define IAGFREELIST_LWM   64

#define INODE_EXTENT_SIZE           IXSIZE      /* inode extent size */
#define NUM_INODE_PER_EXTENT    INOSPEREXT
#define NUM_INODE_PER_IAG           INOSPERIAG

#define MINBLOCKSIZE             512
#define MAXBLOCKSIZE            4096
#define MAXFILESIZE             ((int64_t)1 << 52)

#define JFS_LINK_MAX       65535        /* nlink_t is unsigned short */

/* Minimum number of bytes supported for a JFS partition */
#define MINJFS                  (0x1000000)
#define MINJFSTEXT              "16"

/*
 * fixed physical block address (physical block size = 512 byte)
 *
 * NOTE: since we can't guarantee a physical block size of 512 bytes the use of
 *       these macros should be removed and the byte offset macros used instead.
 */
#define SUPER1_B        64      /* primary superblock */
#define AIMAP_B         (SUPER1_B + 8)  /* 1st extent of aggregate inode map */
#define AITBL_B         (AIMAP_B + 16)  /*
                                         * 1st extent of aggregate inode table
                                         */
#define SUPER2_B        (AITBL_B + 32)  /* 2ndary superblock pbn */
#define BMAP_B          (SUPER2_B + 8)  /* block allocation map */

/*
 * SIZE_OF_SUPER defines the total amount of space reserved on disk for the
 * superblock.  This is not the same as the superblock structure, since all of
 * this space is not currently being used.
 */
#define SIZE_OF_SUPER   PSIZE

/*
 * SIZE_OF_AG_TABLE defines the amount of space reserved to hold the AG table
 */
#define SIZE_OF_AG_TABLE        PSIZE

/*
 * SIZE_OF_MAP_PAGE defines the amount of disk space reserved for each page of
 * the inode allocation map (to hold iag)
 */
#define SIZE_OF_MAP_PAGE        PSIZE

/*
 * fixed byte offset address
 */
#define SUPER1_OFF      0x8000  /* primary superblock */
#define AIMAP_OFF       (SUPER1_OFF + SIZE_OF_SUPER)
                                        /*
                                         * Control page of aggregate inode map
                                         * followed by 1st extent of map
                                         */
#define AITBL_OFF       (AIMAP_OFF + (SIZE_OF_MAP_PAGE << 1))
                                        /*
                                         * 1st extent of aggregate inode table
                                         */
#define SUPER2_OFF      (AITBL_OFF + INODE_EXTENT_SIZE)
                                        /*
                                         * secondary superblock
```

```
                                        */
#define BMAP_OFF        (SUPER2_OFF + SIZE_OF_SUPER)
                                       /*
                                        * block allocation map
                                        */

/*
 * The following macro is used to indicate the number of reserved disk blocks at
 * the front of an aggregate, in terms of physical blocks.  This value is
 * currently defined to be 32K.  This turns out to be the same as the primary
 * superblock's address, since it directly follows the reserved blocks.
 */
#define AGGR_RSVD_BLOCKS        SUPER1_B

/*
 * The following macro is used to indicate the number of reserved bytes at the
 * front of an aggregate.  This value is currently defined to be 32K.  This
 * turns out to be the same as the primary superblock's byte offset, since it
 * directly follows the reserved blocks.
 */
#define AGGR_RSVD_BYTES SUPER1_OFF

/*
 * The following macro defines the byte offset for the first inode extent in
 * the aggregate inode table.  This allows us to find the self inode to find the
 * rest of the table.  Currently this value is 44K.
 */
#define AGGR_INODE_TABLE_START  AITBL_OFF

/*
 *      fixed reserved inode number
 */
/* aggregate inode */
#define AGGR_RESERVED_I 0       /* aggregate inode (reserved) */
#define AGGREGATE_I         1   /* aggregate inode map inode */
#define BMAP_I              2   /* aggregate block allocation map inode */
#define LOG_I               3   /* aggregate inline log inode */
#define BADBLOCK_I          4   /* aggregate bad block inode */
#define FILESYSTEM_I    16      /* 1st/only fileset inode in ait:
                                 * fileset inode map inode
                                 */

/* per fileset inode */
#define FILESET_RSVD_I  0       /* fileset inode (reserved) */
#define FILESET_EXT_I   1       /* fileset inode extension */
#define ROOT_I              2   /* fileset root inode */
#define ACL_I               3   /* fileset ACL inode */

#define FILESET_OBJECT_I 4      /* the first fileset inode available for a file
                                 * or directory or link...
                                 */
#define FIRST_FILESET_INO 16    /* the first aggregate inode which describes
                                 * an inode.  (To fsck this is also the first
                                 * inode in part 2 of the agg inode table.)
                                 */

/*
 *      directory configuration
 */
#define JFS_NAME_MAX    255
#define JFS_PATH_MAX    BPSIZE


/*
 *      file system state (superblock state)
 */
#define FM_CLEAN 0x00000000     /* file system is unmounted and clean */
#define FM_MOUNT 0x00000001     /* file system is mounted cleanly */
#define FM_DIRTY 0x00000002     /* file system was not unmounted and clean
                                 * when mounted or
                                 * commit failure occurred while being mounted:
                                 * fsck() must be run to repair
                                 */
#define FM_LOGREDO 0x00000004   /* log based recovery (logredo()) failed:
                                 * fsck() must be run to repair
                                 */
#define FM_EXTENDFS 0x00000008  /* file system extendfs() in progress */

#endif                          /* _H_JFS_FILSYS */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_IMAP
#define _H_JFS_IMAP

/*
 *      jfs_imap.h: disk inode manager
 */

#define EXTSPERIAG      128     /* number of disk inode extent per iag  */
#define IMAPBLKNO         0     /* lblkno of dinomap within inode map   */
#define SMAPSZ            4     /* number of words per summary map      */
#define EXTSPERSUM       32     /* number of extents per summary map entry */
#define L2EXTSPERSUM  5 /* l2 number of extents per summary map */
#define PGSPERIEXT        4     /* number of 4K pages per dinode extent */
#define MAXIAGS         ((1<<20)−1)     /* maximum number of iags       */
#define MAXAG           128     /* maximum number of allocation groups  */

#define AMAPSIZE    512 /* bytes in the IAG allocation maps */
#define SMAPSIZE     16 /* bytes in the IAG summary maps */

/* convert inode number to iag number */
#define INOTOIAG(ino)   ((ino) >> L2INOSPERIAG)

/* convert iag number to logical block number of the iag page */
#define IAGTOLBLK(iagno,l2nbperpg)      (((iagno) + 1) << (l2nbperpg))

/* get the starting block number of the 4K page of an inode extent
 * that contains ino.
 */
#define INOPBLK(pxd,ino,l2nbperpg)      (addressPXD((pxd)) +            \
        ((((ino) & (INOSPEREXT−1)) >> L2INOSPERPAGE) << (l2nbperpg)))

/*
 *      inode allocation map:
 *
 * inode allocation map consists of
 * . the inode map control page and
 * . inode allocation group pages (per 4096 inodes)
 * which are addressed by standard JFS xtree.
 */
/*
 *      inode allocation group page (per 4096 inodes of an AG)
 */
struct iag {
        int64_t   agstart;                  /* 8: starting block of ag              */
        int32_t   iagnum;                   /* 4: inode allocation group number     */
        int32_t   inofreefwd;             /* 4: ag inode free list forward        */
        int32_t   inofreeback;          /* 4: ag inode free list back           */
        int32_t   extfreefwd;             /* 4: ag inode extent free list forward */
        int32_t   extfreeback;          /* 4: ag inode extent free list back    */
        int32_t   iagfree;                  /* 4: iag free list                     */

        /* summary map: 1 bit per inode extent */
        int32_t   inosmap[SMAPSZ];        /* 16: sum map of mapwords w/ free inodes;
                                           *     note: this indicates free and backed
                                           *     inodes, if the extent is not backed the
                                           *     value will be 1.  if the extent is
                                           *     backed but all inodes are being used the
                                           *     value will be 1.  if the extent is
                                           *     backed but at least one of the inodes is
                                           *     free the value will be 0.
                                           */
        int32_t   extsmap[SMAPSZ];        /* 16: sum map of mapwords w/ free extents */
        int32_t   nfreeinos;              /* 4: number of free inodes             */
        int32_t   nfreeexts;              /* 4: number of free extents            */
        /* (72) */
        uint8_t   pad[1976];              /* 1976: pad to 2048 bytes */
        /* allocation bit map: 1 bit per inode (0 − free, 1 − allocated) */
        uint32_t  wmap[EXTSPERIAG];       /* 512: working allocation map  */
        uint32_t  pmap[EXTSPERIAG];       /* 512: persistent allocation map */
        pxd_t     inoext[EXTSPERIAG];  /* 1024: inode extent addresses */
};                      /* (4096) */

/*
```

```
 *      per AG control information (in inode map control page)
 */
struct iagctl {
        int32_t  inofree;               /* 4: free inode list anchor            */
        int32_t  extfree;               /* 4: free extent list anchor           */
        int32_t  numinos;               /* 4: number of backed inodes           */
        int32_t  numfree;               /* 4: number of free inodes             */
};                      /* (16) */

/*
 *      per fileset/aggregate inode map control page
 */
struct dinomap {
        int32_t   in_freeiag;           /* 4: free iag list anchor     */
        int32_t   in_nextiag;           /* 4: next free iag number     */
        int32_t   in_numinos;           /* 4: num of backed inodes */
        int32_t   in_numfree;           /* 4: num of free backed inodes */
        int32_t   in_nbperiext;     /* 4: num of blocks per inode extent */
        int32_t   in_l2nbperiext;       /* 4: l2 of in_nbperiext */
        int32_t   in_diskblock;     /* 4: for standalone test driver  */
        int32_t   in_maxag;                 /* 4: for standalone test driver  */
        uint8_t   pad[2016];            /* 2016: pad to 2048 */
        struct iagctl  in_agctl[MAXAG]; /* 2048: AG control information */
};                      /* (4096) */

#endif                          /* _H_JFS_IMAP */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_SUPERBLOCK
#define _H_JFS_SUPERBLOCK

#ifndef _H_JFS_LOGMGR
#include <uuid/uuid.h>
#endif

/*
 *      jfs_superblock.h
 */

/*
 * make the magic number something a human could read
 */
#define JFS_MAGIC       "JFS1"   /* Magic word: JFS 1 */

#define JFS_VERSION     2       /* Version number: Version 2 */

#define LV_NAME_SIZE    11      /* MUST BE 11 for OS/2 boot sector */

/*
 *      aggregate superblock
 */
struct superblock {
        char s_magic[4];        /* 4: magic number */
        uint32_t s_version;     /* 4: version number */

        int64_t s_size;         /* 8: aggregate size in hardware/LVM blocks;
                                 * VFS: number of blocks
                                 */
        int32_t s_bsize;        /* 4: aggregate block size in bytes;
                                 * VFS: fragment size
                                 */
        int16_t s_l2bsize;      /* 2: log2 of s_bsize */
        int16_t s_l2bfactor;    /* 2: log2(s_bsize/hardware block size) */
        int32_t s_pbsize;       /* 4: hardware/LVM block size in bytes */
        int16_t s_l2pbsize;     /* 2: log2 of s_pbsize */
        int16_t pad;            /* 2: padding necessary for alignment */

        uint32_t s_agsize;      /* 4: allocation group size in aggr. blocks */

        uint32_t s_flag;        /* 4: aggregate attributes:
                                 *    see jfs_filsys.h
                                 */
        uint32_t s_state;       /* 4: mount/unmount/recovery state:
                                 *    see jfs_filsys.h
                                 */
        int32_t s_compress;     /* 4: > 0 if data compression */

        pxd_t s_ait2;           /* 8: first extent of secondary
                                 *    aggregate inode table
                                 */

        pxd_t s_aim2;           /* 8: first extent of secondary
                                 *    aggregate inode map
                                 */
        uint32_t s_logdev;      /* 4: device address of log */
        int32_t s_logserial;    /* 4: log serial number at aggregate mount */
        pxd_t s_logpxd;         /* 8: inline log extent */

        pxd_t s_fsckpxd;        /* 8: inline fsck work space extent */

        struct timestruc_t s_time;      /* 8: time last updated */

        int32_t s_fsckloglen;   /* 4: Number of filesystem blocks reserved for
                                 *    the fsck service log.
                                 *    N.B. These blocks are divided among the
                                 *         versions kept.  This is not a per
                                 *         version size.
                                 *    N.B. These blocks are included in the
                                 *         length field of s_fsckpxd.
                                 */
        int8_t s_fscklog;       /* 1: which fsck service log is most recent
```

```
                                        *    0 => no service log data yet
                                        *    1 => the first one
                                        *    2 => the 2nd one
                                        */
        char s_fpack[11];               /* 11: file system volume name
                                        *     N.B. This must be 11 bytes to
                                        *          conform with the OS/2 BootSector
                                        *          requirements
                                        *          Only used when s_version is 1
                                        */

        /* extendfs() parameter under s_state & FM_EXTENDFS */
        int64_t s_xsize;                /* 8: extendfs s_size */
        pxd_t s_xfsckpxd;               /* 8: extendfs fsckpxd */
        pxd_t s_xlogpxd;                /* 8: extendfs logpxd */
        /* − 128 byte boundary − */

        uuid_t s_uuid;                  /* 16: 128−bit uuid for volume */
        char s_label[16];               /* 16: volume label */
        uuid_t s_loguuid;               /* 16: 128−bit uuid for log device */
};

#endif /*_H_JFS_SUPERBLOCK */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_XTREE
#define _H_JFS_XTREE

/*
 *      jfs_xtree.h: extent allocation descriptor B+−tree manager
 */

#include "jfs_btree.h"

/*
 *      extent allocation descriptor (xad)
 */
typedef struct xad {
        unsigned flag:8;        /* 1: flag */
        unsigned rsvrd:16;      /* 2: reserved */
        unsigned off1:8;        /* 1: offset in unit of fsblksize */
        uint32_t off2;          /* 4: offset in unit of fsblksize */
        unsigned len:24;        /* 3: length in unit of fsblksize */
        unsigned addr1:8;       /* 1: address in unit of fsblksize */
        uint32_t addr2;         /* 4: address in unit of fsblksize */
} xad_t;                        /* (16) */

#define MAXXLEN         ((1 << 24) − 1)

#define XTSLOTSIZE      16
#define L2XTSLOTSIZE    4

/* xad_t field construction */
#define XADoffset(xad, offset64)\
{\
        (xad)−>off1 = ((uint64_t)offset64) >> 32;\
        (xad)−>off2 = __cpu_to_le32((offset64) & 0xffffffff);\
}
#define XADaddress(xad, address64)\
{\
        (xad)−>addr1 = ((uint64_t)address64) >> 32;\
        (xad)−>addr2 = __cpu_to_le32((address64) & 0xffffffff);\
}
#define XADlength(xad, length32)        (xad)−>len = __cpu_to_le24(length32)

/* xad_t field extraction */
#define offsetXAD(xad)\
        ( ((int64_t)((xad)−>off1)) << 32 | __le32_to_cpu((xad)−>off2))
#define addressXAD(xad)\
        ( ((int64_t)((xad)−>addr1)) << 32 | __le32_to_cpu((xad)−>addr2))
#define lengthXAD(xad)  __le24_to_cpu((xad)−>len)

/* xad_t flags */
#define XAD_NEW         0x01    /* new */
#define XAD_EXTENDED    0x02    /* extended */
#define XAD_COMPRESSED  0x04    /* compressed with recorded length */
#define XAD_NOTRECORDED 0x08    /* allocated but not recorded */
#define XAD_COW         0x10    /* copy−on−write */


/* possible values for maxentry */
#define XTROOTINITSLOT_DIR  6
#define XTROOTINITSLOT     10
#define XTROOTMAXSLOT      18
#define XTPAGEMAXSLOT     256
#define XTENTRYSTART        2

/*
 *      xtree page:
 */
typedef union {
        struct xtheader {
                int64_t next;       /* 8: */
                int64_t prev;       /* 8: */

                uint8_t flag;       /* 1: */
                uint8_t rsrvd1;     /* 1: */
                int16_t nextindex;      /* 2: next index = number of entries */
```

```
91                  int16_t maxentry;       /* 2: max number of entries */
92                  int16_t rsrvd2;     /* 2: */
93
94                  pxd_t   self;       /* 8: self */
95          } header;                       /* (32) */
96
97          xad_t xad[XTPAGEMAXSLOT];       /* 16 * maxentry: xad array */
98  } xtpage_t;
99
100 #endif                          /* !_H_JFS_XTREE */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef  __H_DEBUG__

#define __H_DEBUG__

#include <stdio.h>

#ifdef TRACE
#define DBG_TRACE(a) {printf a; fflush(stdout);}
#else
#define DBG_TRACE(a) ;
#endif

#ifdef TRACE_IO
#define DBG_IO(a) {printf a; fflush(stderr);}
#else
#define DBG_IO(a) ;
#endif

#ifdef TRACE_ERROR
#define DBG_ERROR(a) {printf a; fflush(stdout);}
#else
#define DBG_ERROR(a) ;
#endif

#endif
```

```c
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#include "jfs_types.h"
#include "jfs_dmap.h"
#include "diskmap.h"

/*
 * budtab[]
 *
 * used to determine the maximum free string in a character
 * of a wmap word.  the actual bit values of the character
 * serve as the index into this array and the value of the
 * array at that index is the max free string.
 *
 */
static int8_t budtab[256] = {
        3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, −1
};

/*
 * NAME: ujfs_maxbuddy
 *
 * FUNCTION: Determines the maximum string of free blocks within a word of the
 *      wmap or pmap consistent with binary buddy.
 *
 * PRE CONDITIONS:
 *
 * POST CONDITIONS:
 *
 * PARAMETERS:
 *      cp      − Pointer to wmap or pmap word.
 *
 * NOTES:
 *
 * DATA STRUCTURES:
 *
 * RETURNS: Maximum string of free blocks within word.
 */
int8_t ujfs_maxbuddy(unsigned char *cp)
{
        /*
         * Check if the wmap or pmap word is all free. If so, the free buddy size is
         * BUDMIN.
         */
        if (*((uint32_t *) cp) == 0) {
                return (BUDMIN);
        }

        /*
         * Check if the wmap or pmap word is half free. If so, the free buddy size
         * is BUDMIN−1.
         */
        if (*((uint16_t *) cp) == 0 || *((uint16_t *) cp + 1) == 0) {
                return (BUDMIN − 1);
        }

        /*
         * Not all free or half free. Determine the free buddy size through table
         * lookup using quarters of the wmap or pmap word.
         */
```

```
91          return (MAX(MAX(budtab[*cp], budtab[*(cp + 1)]),
92                      MAX(budtab[*(cp + 2)], budtab[*(cp + 3)])));
93  }
94
95  /*
96   * NAME: ujfs_adjtree
97   *
98   * FUNCTION: Calculate the tree of a dmap or dmapctl.
99   *
100  * PRE CONDITIONS:
101  *
102  * POST CONDITIONS:
103  *
104  * PARAMETERS:
105  *       cp      − Pointer to the top of the tree
106  *       l2leaves− Number of leaf nodes as a power of 2
107  *       l2min   − Number of disk blocks actually covered by a leaf of the tree;
108  *                 specified as a power of 2
109  *
110  * NOTES: This routine first works against the leaves of the tree to calculate
111  *       the maximum free string for leaf buddys.  Once this is accomplished the
112  *       values of the leaf nodes are bubbled up the tree.
113  *
114  * DATA STRUCTURES:
115  *
116  * RETURNS:
117  */
118 int8_t ujfs_adjtree(int8_t * treep, int32_t l2leaves, int32_t l2min)
119 {
120         int32_t nleaves, leaf_index, l2max, nextb, bsize, index;
121         int32_t l2free, leaf, num_this_level, nextp;
122         int8_t *cp0, *cp1, *cp = treep;
123
124         /*
125          * Determine the number of leaves of the tree and the
126          * index of the first leaf.
127          * Note: I don't know why the leaf_index calculation works, but it does.
128          */
129         nleaves = (1 << l2leaves);
130         leaf_index = (nleaves − 1) / 3;
131
132         /*
133          * Determine the maximum free string possible for the leaves.
134          */
135         l2max = l2min + l2leaves;
136
137         /*
138          * Try to combine buddies starting with a buddy size of 1 (i.e. two leaves).
139          * At a buddy size of 1 two buddy leaves can be combined if both buddies
140          * have a maximum free of l2min; the combination will result in the
141          * left−most buddy leaf having a maximum free of l2min+1.  After processing
142          * all buddies for a certain size, process buddies at the next higher buddy
143          * size (i.e. current size * 2) and the next maximum free
144          * (current free + 1).  This continues until the maximum possible buddy
145          * combination yields maximum free.
146          */
147         for (l2free = l2min, bsize = 1; l2free < l2max; l2free++, bsize = nextb) {
148                 nextb = bsize << 1;
149
150                 for (cp0 = cp + leaf_index, index = 0; index < nleaves;
151                      index += nextb, cp0 += nextb) {
152                    if (*cp0 == l2free && *(cp0 + bsize) == l2free) {
153                            *cp0 = l2free + 1;
154                            *(cp0 + bsize) = −1;
155                    }
156                 }
157         }
158
159         /*
160          * With the leaves reflecting maximum free values bubble this information up
161          * the tree.  Starting at the leaf node level, the four nodes described by
162          * the higher level parent node are compared for a maximum free and this
163          * maximum becomes the value of the parent node.  All lower level nodes are
164          * processed in this fashion then we move up to the next level (parent
165          * becomes a lower level node) and continue the process for that level.
166          */
167         for (leaf = leaf_index, num_this_level = nleaves >> 2; num_this_level > 0;
168              num_this_level >>= 2, leaf = nextp) {
169                 nextp = (leaf − 1) >> 2;
170
171                 /*
172                  * Process all lower level nodes at this level setting the value of the
173                  * parent node as the maximum of the four lower level nodes.
174                  */
175                 for (cp0 = cp + leaf, cp1 = cp + nextp, index = 0;
176                      index < num_this_level; index++, cp0 += 4, cp1++) {
177                         *cp1 = TREEMAX(cp0);
178                 }
179         }
180
```

```
181         return (*cp);
182 }
183
184 /*
185  * NAME: ujfs_complete_dmap
186  *
187  * FUNCTION: Fill in rest of dmap fields from wmap/pmap already initialized.
188  *
189  * PARAMETERS:
190  *      dmap_page       − dmap to complete
191  *      blkno           − starting block number for this dmap
192  *      treemax         − will be filled in with max free for this dmap
193  *
194  * RETURNS: NONE
195  */
196 void ujfs_complete_dmap(struct dmap *dmap_page, int64_t blkno, int8_t *treemax)
197 {
198         struct dmaptree *tp;
199         int8_t *cp;
200         int32_t index;
201
202         dmap_page−>start = blkno;
203
204         tp = &dmap_page−>tree;
205         tp−>height = 4;
206         tp−>leafidx = LEAFIND;
207         tp−>nleafs = LPERDMAP;
208         tp−>l2nleafs = L2LPERDMAP;
209         tp−>budmin = BUDMIN;
210
211         /*
212          * Pick up the pointer to the first leaf of the dmap tree.
213          */
214         cp = tp−>stree + tp−>leafidx;
215
216         /*
217          * Set the initial state for the leaves of the dmap tree.  They will reflect
218          * the current allocation state of the wmap words.
219          */
220         for (index = 0; index < LPERDMAP; index++) {
221                 *(cp + index) = ujfs_maxbuddy((unsigned char *) &dmap_page−>wmap[index]);
222         }
223
224         /*
225          * With the leaves of the dmap initialized adjust (initialize) the dmap's
226          * tree.
227          */
228         *treemax = ujfs_adjtree(tp−>stree, L2LPERDMAP, BUDMIN);
229 }
230
231 /*
232  * NAME: ujfs_idmap_page
233  *
234  * FUNCTION: Initialize one dmap page
235  *
236  * POST CONDITIONS: Blocks which don't actually exist in the aggregate will be
237  *      marked as allocated in the dmap page.  The total number of blocks will
238  *      only account for the actually existing blocks.
239  *
240  * PARAMETERS:
241  *      map_page        − pointer to page of map
242  *      nblocks         − number of blocks this page
243  *
244  * RETURNS: NONE
245  */
246 void ujfs_idmap_page(struct dmap *map_page, uint32_t nblocks)
247 {
248         uint32_t index, nwords, bit;
249
250         map_page−>nblocks = map_page−>nfree = nblocks;
251
252         /*
253          * Partial dmap page?
254          * If there are not enough blocks to cover an entire dmap page the ones
255          * which represent blocks which don't exist will be marked as allocated.
256          *
257          * nwords will indicate the first word beyond the end of existing blocks
258          * bit will indicate if this block does not fall on a 32−bit boundary
259          */
260         nwords = nblocks / DBWORD;
261         bit = nblocks % DBWORD;
262
263         if (bit) {
264                 /*
265                  * Need to mark a partial word allocated
266                  */
267                 map_page−>wmap[nwords] = map_page−>pmap[nwords] = ONES >> bit;
268                 nwords++;
269         }
270
```

```
271             /*
272              * Set the rest of the words in the page to ONES.
273              */
274             for (index = nwords; index < LPERDMAP; index++) {
275                     map_page−>pmap[index] = map_page−>wmap[index] = ONES;
276             }
277     }
278
279     /*
280      * NAME: ujfs_getagl2size
281      *
282      * FUNCTION: Determine log2(allocation group size) based on size of aggregate
283      *
284      * PARAMETERS:
285      *      size            − Number of blocks in aggregate
286      *      aggr_block_size − Aggregate block size
287      *
288      * RETURNS: log2(allocation group size) in aggregate blocks
289      */
290     int32_t ujfs_getagl2size(int64_t size, int32_t aggr_block_size)
291     {
292             int64_t sz;
293             int64_t m;
294             int32_t l2sz;
295
296             if (size < BPERDMAP * MAXAG) {
297                     return (L2BPERDMAP);
298             }
299
300             m = ((uint64_t) 1 << (64 − 1));
301             for (l2sz = 64; l2sz >= 0; l2sz−−, m >>= 1) {
302                     if (m & size) {
303                             break;
304                     }
305             }
306
307             sz = (int64_t) 1 << l2sz;
308             if (sz < size) {
309                     l2sz += 1;
310             }
311
312             return (l2sz − L2MAXAG);
313     }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_DISKMAP
#define H_DISKMAP

#include <jfs_types.h>          /* int8_t, int32_t, int64_t, uint32_t */

int8_t ujfs_maxbuddy(unsigned char *);
int8_t ujfs_adjtree(int8_t *, int32_t, int32_t);
void ujfs_complete_dmap(struct dmap *, int64_t, int8_t *);
void ujfs_idmap_page(struct dmap *, uint32_t);
int32_t ujfs_getagl2size(int64_t, int32_t);

#endif                          /* H_DISKMAP */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _FSCK_BASE_H_
#define _FSCK_BASE_H_

/*
   −−−−−−−−− Defines shared among the fsck modules −−−−−−−−−
*/

#define ISDIR(m)         (((m)&(IFMT)) == (IFDIR))
#define ISREG(m)         (((m)&(IFMT)) == (IFREG))
#define ISLNK(m)         (((m)&(IFMT)) == (IFLNK))
#define ISBLK(m)         (((m)&(IFMT)) == (IFBLK))
#define ISCHR(m)         (((m)&(IFMT)) == (IFCHR))
#define ISFIFO(m)        (((m)&(IFMT)) == (IFFIFO))
#define ISSOCK(m)        (((m)&(IFMT)) == (IFSOCK))

/* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
 * some useful constants
 */
#define BYTESPERPAGE      4096
#define log2BYTESPERPAGE  12
#define BITSPERPAGE       (4096*8)
#define log2BITSPERPAGE   15
#define BITSPERDWORD      32
#define log2BITSPERDWORD  5
#define BYTESPERDWORD     4
#define log2BYTESPERDWORD 2
#define BITSPERBYTE       8
#define log2BITSPERBYTE   3
#define MEMSEGSIZE        (64*1024)
#define log2BYTESPERKBYTE 10

#endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_JFS_FSCKCBBL
#define _H_JFS_FSCKCBBL

/*
 *  This structure resides in the first page (aka the block map control page)
 *  of the fsck in−aggregate workspace.  JFS Clear Bad Block List utility
 *  processing writes to this record, then fsck reads it and reports to the
 *  caller.
 */
struct fsckcbbl_record {
        char eyecatcher[8];
        char avail_1[4];        /*   4 */
        int32_t cbbl_retcode;   /* JFS Clear Bad Block List utility
                                   * return code
                                 */
        int32_t fs_blksize;     /* aggregate block size */
        int32_t lv_blksize;     /* device block size    */
        int32_t fs_lv_ratio;    /* fs_blksize/lv_blksize */
        int64_t fs_last_metablk;        /*
                                         * last fs block we won't try to relocate
                                         * because it holds fixed−location metadata
                                         */
        int64_t fs_first_wspblk;        /*
                                         * first fs block we won't try to relocate
                                         * because it holds fsck workspace or the
                                         * inline journal log
                                         */
        int32_t total_bad_blocks;       /* count of bad blocks in LVM's list
                                         * at beginning of Bad Block List utility
                                         * processing
                                         */
        int32_t resolved_blocks;        /* count of bad blocks:
                                         * − for which the data has been relocated,
                                         * − which are now allocated to the bad block
                                         *   inode, and
                                         * − which the LVM has been told to forget
                                         */
        int32_t reloc_extents;  /* count of relocated extents */
        int64_t reloc_blocks;   /* count of blocks in relocated extents */
        int32_t LVM_lists;      /* count of bad block lists maintained by LVM
                                   * according to the last query
                                 */
        char bufptr_eyecatcher[8];
        void *clrbblks_agg_recptr;      /* addr of clrbblks aggregate record   */
        void *ImapInoPtr;       /* addr of imap inode buffer         */
        void *ImapCtlPtr;       /* addr of imap control page buffer  */
        void *ImapLeafPtr;      /* addr of imap leaf page buffer     */
        void *iagPtr;           /* addr of iag buffer                        */
        void *InoExtPtr;        /* addr of inode extent buffer               */
        char avail_2[28];       /*   28 */
};                              /* total = 128 bytes */

#endif                          /*  _H_JFS_FSCKCBBL */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_FSCKLOG
#define H_FSCKLOG

#include <jfs_types.h>

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 *
 * The in−aggregate fsck service log
 */

#define flog_eyecatcher_string "fscklog "

struct fscklog_entry_hdr {
        int16_t entry_length;
        int16_t msg_num;
        int16_t num_parms;
};

struct fscklog_error {
        int64_t err_offset;     /*  8 −− start of attempted write */
        int32_t bytes_written;  /*  4 −− number actually written  */
        int32_t io_retcode;     /*  4 −− return code from write   */
};                              /*    total: 16 bytes */

/* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
 *
 *  The first log message number for the JFS fsck Service Log
 *
 *
 */

#define JFSCHKLOG_FIRSTMSGNUM 10000

/* + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
 *
 *  The format of the log entry in the file created by
 *  extracting the contents of the fsck service log
 *  (stored in the fsck workspace) into a file.
 *
 *
 * N.B. The log entries are blocked such that no log entry crosses
 *      an XCHKLOG_BUFSIZE byte boundary in the file.
 */

#define XCHKLOG_BUFSIZE  8192

#define jfs_chklog_eyecatcher "JFS chkdskSvcLog"

struct chklog_entry_hdr {
        int16_t entry_length;
        int16_t msg_num;
        int16_t num_parms;
};

#endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_FSCKWSP
#define H_FSCKWSP

#include "fscklog.h"
#include <jfs_dmap.h>
#include <jfs_dtree.h>
#include <jfs_xtree.h>
#include <jfs_filsys.h>
#include <jfs_imap.h>
#include <jfs_dinode.h>
#include "fsck_base.h"
#include "fsckcbbl.h"

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 *
 * fsck uses the structures and types in this file to define its workspace.
 *
 *++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 */

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * type dtree_Q_elem describes one node which is to be visited in the
 *                  traversal of a directory B+ Tree
 *
 */
struct dtreeQelem {
        struct dtreeQelem *next;
        struct dtreeQelem *prev;
        int64_t node_addr;
        pxd_t node_pxd;
        uint8_t node_key_len;
        UniChar node_key[JFS_NAME_MAX];
        int8_t node_level;
        uint32_t node_size;
};

#define dtreeQ_elem_length sizeof(struct dtreeQelem)

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * type tree_Q_elem describes one node which is to be visited in the
 *                  traversal of a non−directory B+ Tree
 *
 */
struct treeQelem {
        struct treeQelem *next;
        struct treeQelem *prev;
        int64_t node_addr;
        int64_t node_first_offset;
        pxd_t node_pxd;
        int8_t node_level;
        int8_t last_ext_uneven;
};

#define treeQ_elem_length sizeof(struct treeQelem)

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * type dupall_blkrec describes one aggregate block which has been found to
 *                    have more than 1 inode claiming to own it.
 *
 */
struct dupall_blkrec {
        int64_t blk_number;
        uint8_t first_ref_resolved;
        uint8_t avail_byte;
        uint16_t owner_count;
        struct dupall_blkrec *next;
};

#define dupall_blkrec_length sizeof(struct dupall_blkrec)

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 *
 * Each (4096 byte) page in the block map describes 32768
```

```
 * blocks in the aggregate.
 */

/*
 * the fsck Workspace Block Map control page
 *
 */

struct fsck_blk_map_hdr {
        /*
         * the 1st 1024 bytes are the clrbblks to fsck
         * communication area
         */
        struct fsckcbbl_record cbblrec; /* 128 */
        char avail_0[896];      /* 896 */
        /* 1024 byte boundary */
        /*
         * the next 2048 bytes are the fsck block map
         * header area
         */
        struct {
                char eyecatcher[8];             /*    8 */
                int32_t last_entry_pos;         /*    4 */
                int32_t next_entry_pos;         /*    4 */
                char start_time[20];            /*   20 */
                char end_time[20];              /*   20 */
                int32_t return_code;            /*    4 */
                char *super_buff_addr;          /*    4 -- for the dump reader */
                char *agg_record_addr;          /*    4 -- for the dump reader */
                char *bmap_record_addr;         /*    4 -- for the dump reader */
                char avail_1[8];                /*    8 */
                int64_t fscklog_agg_offset;     /*    8 */
                int8_t fscklog_full;            /*    1 */
                int8_t fscklog_buf_allocated;   /*    1 */
                int8_t fscklog_buf_alloc_err;   /*    1 */
                char avail_2[1];                /*    1 */
                int32_t num_logwrite_errors;    /*    4 */
                char avail_3[32];               /*   32 */
                struct fscklog_error logerr[125];       /* 1920 = 120 * 16 */
        } hdr;                                  /* subtotal, 2048 bytes */
        /*
         * the last 1024 bytes are available
         */
        char avail_5[1024];     /* 1024 */
};                              /* total: 4096 bytes */

#define fbmh_eyecatcher_string "wspblkmp"

/*
 * a page in the fsck Workspace Block Map
 */
struct fsck_blk_map_page {
        uint32_t fsck_blkmap_words[1024];       /* 4096 bytes */
};

/*--------------------------------------------------------------------
 * structure blkmap_wspace describes the portion of the aggregate
 *                          record allocated for use when verifying
 *                          the aggregate block map.
 *
 */
struct blkmap_wspace {
        uint32_t dmap_map[LPERDMAP];    /* bit map of blk allocations */
        int8_t dmap_wsp_tree[TREESIZE - LPERDMAP];
        int8_t dmap_wsp_leafs[LPERDMAP];
        int8_t L0_wsp_tree[CTLTREESIZE - LPERCTL];
        int8_t L0_wsp_leafs[LPERCTL];
        int8_t L1_wsp_tree[CTLTREESIZE - LPERCTL];
        int8_t L1_wsp_leafs[LPERCTL];
        int8_t L2_wsp_tree[CTLTREESIZE - LPERCTL];
        int8_t L2_wsp_leafs[LPERCTL];
        int64_t AG_free[MAXAG];
};

/*--------------------------------------------------------------------
 * structure fsck_iag_record describes one Inode Allocation
 *                            Group belonging to an inode table in
 *                            the aggregate
 *
 */
struct fsck_iag_record {
        uint32_t AG_num;
        uint32_t backed_inodes;
        uint32_t unused_backed_inodes;
        uint32_t unbacked_extents;
};

#define iag_record_length sizeof(struct fsck_iag_record)

/*--------------------------------------------------------------------
```

```
 * structure fsck_ag_record describes one Allocation Group
 *                           as it relates to an inode table in the
 *                           aggregate
 *
 */
struct fsck_ag_record {
        uint32_t backed_inodes;
        uint32_t unused_backed_inodes;
        int32_t frext_list_first;
        int32_t frext_list_last;
        int32_t frext_list_len;
        int32_t frino_list_first;
        int32_t frino_list_last;
        int32_t frino_list_len;
        int8_t frext_list_bad;
        int8_t frino_list_bad;
        char unused[14];
};

#define ag_record_length sizeof(struct fsck_ag_record)

/*--------------------------------------------------------------------
 * structure fsck_iam_record describes one Inode Allocation Map
 *                           in the aggregate
 *
 */
struct fsck_iam_record {
        int64_t first_leaf_offset;      /* offset of first leaf for imap inode */
        int8_t imap_is_rootleaf;
        int8_t friag_list_bad;
        char unused[2];
        int32_t bkd_inodes;     /* count of backed inodes    */
        int32_t unused_bkd_inodes;      /* count of available backed inodes */
        int32_t num_iags;       /* number of IAGs in the inode tbl */
        int32_t friag_list_last;
        int32_t friag_list_len;
        struct fsck_iag_record *iag_tbl;        /* ptr to info by iag */
        struct fsck_ag_record *ag_tbl;  /* ptr to info by ag  */
};

#define iam_record_length sizeof(struct fsck_iam_record)

/*--------------------------------------------------------------------
 * structure fsck_inode_record describes one inode belonging to
 *                             the aggregate or to a fileset
 *                             in the aggregate
 *
 */

/*
 * the possible values for inode_type
 *   (since inode_type is 4 bits wide, valid range is 0−15)
 */
#define unused_inode        0
#define file_inode          1
#define strm_descr_inode    2
#define stream_inode        3
#define directory_inode     4
#define symlink_inode       5
#define block_special_inode 6
#define char_special_inode  7
#define FIFO_inode          8
#define SOCK_inode          9
#define metadata_inode     14
#define unrecognized_inode 15

#define add_direntry_extension 'A'
#define rmv_direntry_extension 'R'
#define rmv_badentry_extension 'B'
#define parent_extension 'P'

struct fsck_inode_ext_record {
        char ext_type;          /* 1  : extens type (see constants above) */
        unsigned ino_type:4;    /*  .5: see constants above
                                 *      describes the inonum inode.
                                 *      This is only interesting when
                                 *      type is rmv_direntry_extension.
                                 */
        unsigned unused_byte:20;        /* 2.5: unused      */
        struct fsck_inode_ext_record *next;     /*
                                                 * 4  : addr next extension record
                                                 */
        uint32_t inonum;        /* 4  : inode number */
};

/*
 * the inode record data
 */
struct fsck_inode_record {
        uint32_t inonum;        /* 4.0 : key         */
```

```
271         /* 3 : 24 flag bits */
272         unsigned in_use:1;
273         unsigned selected_to_rls:1;
274         unsigned crrct_link_count:1;
275         unsigned crrct_prnt_inonum:1;
276
277         unsigned adj_entries:1;
278         unsigned cant_chkea:1;
279         unsigned clr_ea_fld:1;
280         unsigned clr_acl_fld:1;
281
282         unsigned ignore_ea_blks:1;
283         unsigned ignore_acl_blks:1;
284         unsigned ignore_alloc_blks:1;
285         unsigned inline_data_err:1;
286
287         unsigned reconnect:1;
288         unsigned unxpctd_prnts:1;
289         unsigned badblk_inode:1;
290         unsigned involved_in_dups:1;
291
292         unsigned crrct_cumm_blks:1;
293         unsigned avail_flagbit_1:1;
294         unsigned inlineea_on:1;
295         unsigned inlineea_off:1;
296
297         unsigned avail_flagbits:4;
298         /*        end of flag bits          */
299         int8_t inode_type;      /* 1 : see constants above          */
300
301         int32_t link_count;     /* 4 : Stored link count is added to this.
302                                  *    Decremented for each observed link.
303                                  */
304         uint32_t parent_inonum; /* 4 : If this is a directory inode and
305                                  * the unxpctd_prnts bit is set, then
306                                  * this field contains the inode number
307                                  * stored in the parent field on disk.
308                                  * Otherwise, this is the inode number
309                                  * of the first observed parent.
310                                  */
311         int64_t cumm_blocks;    /* 8 : number of blocks allocated to the
312                                  * inode and, if this is a directory, to each
313                                  * inode described in an entry
314                                  */
315         int32_t dtree_level;    /* 4 : distance, in the fileset directory tree,
316                                  * from the root (inode 2)
317                                  */
318         struct fsck_inode_ext_record *ext_rec;  /* 4.0: extension record address         */
319 };
320
321 #define inode_record_length sizeof(struct fsck_inode_record)
322
323 #define inode_is_metadata( X ) \
324              ( (int8_t) ((X)->inode_type) == (int8_t) metadata_inode )
325
326 /*--------------------------------------------------------------------------
327  * type inode_tbl_t is an array of pointers to inode records
328  *
329  * There is one of these tables for each allocated inode extent which
330  * has at least one inode in use.
331  *
332  */
333 struct inode_tbl_t {
334         char eyecatcher[8];     /* 8.0 */
335         struct fsck_inode_record *inorectbl[32];        /* 128 */
336 };
337
338 #define inode_tbl_length sizeof( struct inode_tbl_t )
339
340 /*--------------------------------------------------------------------------
341  * type inode_ext_tbl_t is an array of pointers to inode tables
342  *
343  * There is one of these tables for each allocated IAG which
344  * has at least one allocated extent with an inode in use.
345  *
346  */
347 struct inode_ext_tbl_t {
348         char eyecatcher[8];     /* 8.0 */
349         struct inode_tbl_t *inotbl[128];        /* 512 */
350 };
351
352 #define inode_ext_tbl_length sizeof( struct inode_ext_tbl_t )
353
354 /*--------------------------------------------------------------------------
355  * type IAG_tbl_t is an array of pointers to inode extent tables
356  *
357  * There is one of these tables for each fileset in the aggregate,
358  * and is allocated dynamically when the number of IAGs in the
359  * fileset Inode Map is known.
360  *
```

```
 */
struct IAG_tbl_t {
        char eyecatcher[8];     /* 8.0 */
        struct inode_ext_tbl_t *inoext_tbl[1];  /* 4.0 */
};

/*--------------------------------------------------------------------------
 * type wsp_ext_rec describes one extent of storage which fsck
 *                          has allocated for its workspace.
 *
 */
struct wsp_ext_rec {
        struct wsp_ext_rec *next;
        int8_t from_high_memory;
        int8_t for_logredo;
        char avail[2];
        uint32_t extent_length;
        char *extent_addr;
        uint32_t last_byte_used;
};

/*--------------------------------------------------------------------------
 * type recon_buf_record   describes one dnode I/O buffer used during
 *                         inode reconnect processing
 *
 */
struct recon_buf_record {
        dtpage_t dnode_buf;
        struct recon_buf_record *stack_next;
        int64_t dnode_blkoff;
        int64_t dnode_byteoff;
        char reserved[12];
};

/*--------------------------------------------------------------------------
 * type treeStack_record describes one stack element used during DASD
 *                          limits processing
 *
 */
struct treeStack_record {
        struct fsck_inode_record *inorec;
        struct treeStack_record *next;
};

#define treeStack_elem_length sizeof(struct treeStack_record)

/*--------------------------------------------------------------------------
 * structure fsck_agg_record describes the aggregate
 */

struct fsck_agg_record {
        char eyecatcher[8];
        uint32_t ondev_jlog_fsblk_length;
        int64_t ondev_jlog_fsblk_offset;
        int64_t ondev_wsp_byte_length;
        int64_t ondev_wsp_byte_offset;
        uint32_t ondev_wsp_fsblk_length;
        int64_t ondev_wsp_fsblk_offset;
        uint32_t ondev_fscklog_fsblk_length;
        int64_t ondev_fscklog_fsblk_offset;
        int64_t ondev_fscklog_byte_length;
        int64_t ondev_fscklog_byte_offset;
        uint32_t log2_blksize;
        uint32_t blksperpg;
        uint32_t log2_blksperpg;
        uint32_t log2_blksperag;
        int64_t sb_agg_fsblk_length;
        uint32_t inode_stamp;   /* taken from the agg self inode, every
                                 * in-use, valid inode must have a
                                 * matching di_inostamp.
                                 */
        int inode_count;        /* num inodes owned by the aggregate */
        int inodes_inuse;       /* num aggregate nodes now in use    */
        int fset_inodes_inuse;  /* num fileset inodes in use         */
        int fset_inodes_seen;   /* num fileset inodes scanned.       */
        int fset_inode_count;   /* num inodes in the filesets        */
        int64_t dup_block_count;        /* num multiply-allocated
                                         * blocks seen
                                         */
        int64_t unresolved_1stref_count;        /* num unresolved first
                                                 * references to multiply allocated
                                                 * blocks
                                                 */
        char aggrec_rsvd0[4];
        int64_t blocks_for_inodes;      /* by fsck's count           */
        int64_t blocks_for_files;       /* by fsck's count           */
        int64_t blocks_for_dirs;        /* by fsck's count           */
        int64_t blocks_for_eas;         /* by fsck's count           */
        int64_t blocks_for_acls;        /* by fsck's count           */
        uint64_t inodes_in_aggregate;   /* by fsck's count           */
```

```
451         uint64_t files_in_aggregate;    /* by fsck's count            */
452         uint64_t dirs_in_aggregate;     /* by fsck's count            */
453         int64_t free_blocks_in_aggregate;       /* by fsck's count            */
454         int64_t blocks_used_in_aggregate;       /* by fsck's count            */
455         uint64_t blocks_this_fset;      /* by fsck's count            */
456
457         int32_t logredo_rc;     /* logredo return code              */
458         int32_t logformat_rc;   /* logformat return code            */
459         char aggrec_rsvd2[8];
460
461         struct {                /* data for the current inode          */
462                 char eyecatcher[8];
463                 int64_t all_blks;       /* all blocks allocated               */
464                 int64_t data_blks;      /* blocks allocated to data           */
465                 uint64_t data_size;     /* byte offset of last data if file   */
466                 int64_t ea_blks;        /* blocks allocated to EAs            */
467                 int64_t acl_blks;       /* blocks allocated to ACLs           */
468                 uint16_t ea_inline;
469                 uint16_t inline_ea_offset;
470                 uint16_t inline_ea_length;
471                 uint16_t acl_inline;
472                 uint16_t inline_acl_offset;
473                 uint16_t inline_acl_length;
474                 uint16_t in_inode_data_length;  /* length of inline data or of
475                                                  * xad list in the inode
476                                                  */
477                 char aggrec_rsvd4[2];
478         } this_inode;
479
480         struct wsp_ext_rec *recon_buf_extent;
481         struct recon_buf_record *recon_buf_stack;
482
483         char agg_imap_eyecatcher[8];
484         struct fsck_iam_record agg_imap;        /* describes the aggregate imap    */
485
486         char fset_imap_eyecatcher[8];
487         struct fsck_iam_record fset_imap;       /* describes the fileset imap      */
488
489         uint32_t num_ag;        /* number of AGs in the aggregate  */
490         int ag_blk_size;        /* aggregate block size            */
491
492         short int aggrec_rsvd6[1];
493         char delim_char;        /* path delimiter char             */
494         /* unicharacter equivalents                                        */
495         UniChar *UniChar_lsfn_name;     /* equiv to lost+found             */
496         UniChar *UniChar_LSFN_NAME;     /* equiv to LOST+FOUND             */
497         /* end of unicharacter equivalents                                 */
498         int64_t lsfn_dasdused_adjustment;
499         uint32_t lsfn_inonum;   /* inode number of /lost+found/    */
500         uint32_t avail_inonum;  /* inode number of an inode
501                                  * observed to be allocated and not
502                                  * in use (if any)
503                                  */
504         short int parm_options[16];     /* parms specified                 */
505         char aggrec_rsvd7[2];   /* pad to quadword boundary        */
506         char effective_msg_level;
507         char aggrec_rsvd8[7];   /* pad to quadword boundary        */
508         char flags_eyecatcher[8];
509         /* 4 bytes of flags   */
510         unsigned parm_options_nologredo:1;      /* first byte, first nibble   */
511         unsigned processing_readonly:1;
512         unsigned processing_readwrite:1;
513         unsigned messaging_verbose:1;
514         unsigned superblk_ok:1; /* first byte, second nibble  */
515         unsigned aggregate_is_mounted:1;
516         unsigned ag_modified:1;
517         unsigned ag_dirty:1;
518         unsigned lsfn_ok:1;     /* second byte, first nibble  */
519         unsigned cant_write_primary_sb:1;
520         unsigned cant_write_secondary_sb:1;
521         unsigned cant_write_primary_ait:1;
522         unsigned nonmeta_write_error_msg:1;     /* second byte, second nibble */
523         unsigned fsck_is_done:1;
524         unsigned device_is_open:1;
525         unsigned device_is_locked:1;
526         unsigned primary_ait_4part1:1;  /* third byte, first nibble   */
527         unsigned primary_ait_4part2:1;
528         unsigned ait_aim_update_failed:1;
529         unsigned corrections_needed:1;
530         unsigned corrections_approved:1;        /* third byte, second nibble  */
531         unsigned avail_inode_found:1;
532         unsigned initializing_fscklog:1;
533         unsigned warning_pending:1;
534         unsigned high_mem_allocated:1;  /* fourth byte, first nibble  */
535         unsigned rootdir_rebuilt:1;
536         unsigned active_dasd_limits:1;
537         unsigned stdout_redirected:1;
538         unsigned prime_dasd_limits:1;   /* fourth byte, second nibble */
539         unsigned unused_4_2:3;  /*                                 */
540         /* end of flag bytes  */
```

```
541         int path_buffer_length; /* length of path_buffer      */
542         char *path_buffer;      /* storage allocated (if any) for
543                                  * assembling the string containing
544                                  * an inode's path for a message
545                                  */
546         xtpage_t *prim_nodeptr; /* storage allocated (if any) for
547                                  * an xtpage of an inode in the Primary Agg
548                                  * Inode table
549                                  */
550         xtpage_t *second_nodeptr;       /* storage allocated (if any) for
551                                          * an xtpage of an inode in the Secondary
552                                          * Agg Inode table
553                                          */
554         int64_t lowest_valid_fset_datablk;      /* the lowest (aggregate
555                                                  * blocksized) block in the aggregate
556                                                  * AFTER the blocks used for initial
557                                                  * metadata
558                                                  */
559         int64_t highest_valid_fset_datablk;     /* the highest (aggregate
560                                                  * blocksized) block in the aggregate before
561                                                  * the beginning of the fsck workspace
562                                                  */
563         struct fsck_inode_ext_record *free_inode_extens;        /* available workspace inode record
564                                                                  * extensions.
565                                                                  */
566         struct fsck_inode_ext_record *inode_reconn_extens;      /* list of inode record
567                                                                  * extensions describing the directory
568                                                                  * entries to be added when reconnecting
569                                                                  * inodes
570                                                                  */
571         struct treeStack_record *treeStack;     /* head of the stack used during tree
572                                                  * traversal
573                                                  */
574         struct treeStack_record *free_treeStack;        /* available tree stack elements      */
575         int32_t tree_height;
576         struct dtreeQelem *dtreeQ_front;
577         struct dtreeQelem *dtreeQ_back;
578         struct dtreeQelem *free_dtreeQ; /* available dtree stack elements        */
579         struct treeQelem *treeQ_front;
580         struct treeQelem *treeQ_back;
581         struct treeQelem *free_treeQ;   /* available tree stack elements         */
582         struct dupall_blkrec *dup_alloc_lst;    /* list of multiply allocated blocks  */
583         struct dupall_blkrec *free_dupall_blkrec;       /* free dupall_blkrec records    */
584         struct wsp_ext_rec *wsp_extent_list;    /* linked list of records describing
585                                                  * the workspace extents.
586                                                  */
587         char aggrec_rsvda[4];   /* pad to quadword boundary */
588
589         char AIT_eyecatcher[8]; /*                                    */
590         struct inode_tbl_t *AIT_ext0_tbl;       /* ptr to table for Agg Inode Extent 0 */
591         int32_t agg_last_inoidx;        /* used for find first, find next  */
592
593         char FSIT_eyecatcher[8];
594         struct IAG_tbl_t *FSIT_IAG_tbl; /* ptr to table for FSet IAGs      */
595         int32_t fs_last_iagidx; /* used for find first, find next  */
596         int32_t fs_last_extidx; /* used for find first, find next  */
597         int32_t fs_last_inoidx; /* used for find first, find next  */
598
599         struct {                /* data for accessing all allocated inodes in an
600                                  * inode table sequentially
601                                  */
602                 char eyecatcher[8];
603                 int32_t this_iagnum;    /* ordinal of the current iag      */
604                 uint32_t this_inoidx;   /* ordinal of the current inode    */
605                 uint32_t rootleaf_imap; /* 0 => not a rootleaf imap        */
606                 xtpage_t *this_mapleaf; /* current mapleaf in buffer       */
607                 int32_t iagidx_now;     /* index into current mapleaf      */
608                 int32_t iagidx_max;
609                 struct iag *iagptr;     /* current iag in buffer           */
610                 uint32_t extidx_now;    /* index into current iag          */
611                 uint32_t extidx_max;
612                 struct dinode *extptr;  /* current inode extent in buffer  */
613                 uint32_t inoidx_now;    /* index into current inode extent */
614                 uint32_t inoidx_max;
615                 struct dinode *inoptr;  /* current inode in buffer         */
616         } fais;                 /* (For Allocated Inodes Sequentially */
617
618         char aggrec_rsvde[4];   /* pad to quadword boundary */
619
620         /*
621          * The fsck I/O buffer information is below.  Certain information
622          * is kept for each of the buffers:
623          *
624          *    %_buf_ptr      = buffer address
625          *    %_buf_length   = buffer length
626          *    %_buf_data_len = length of data read into buffer at last read
627          *    %_agg_offset   = byte offset in aggregate of buffer contents
628          *    %_buf_write    = flag: !0 => buffer contents modified since
629          *                                 last write to device
630          *
```

```
631           * where  % = {blkmp, iag, mapleaf, ino, node, wsp}
632           *
633           *
634           * N.B. To ensure proper boundary alignment, each IO buffersize
635           *      must be an even number of pages (i.e. 4096 byte pages).
636           *
637           */
638
639          /* ****** very large, multi purpose BUFFER ***** */
640
641  #define VLARGE_BUFSIZE (2 * MEMSEGSIZE)
642
643  #define NOT_CURRENTLY_USED  0
644  #define USED_FOR_EA_BUF  1              /* phase 1 late           */
645  #define USED_FOR_DIRPAGE_BUFS  2        /* phase 6                */
646  #define USED_FOR_INOEXT_BUF  3          /* phase 1 early, phase 7 */
647  #define USED_FOR_SUPER_VALIDATION 4     /* initial processing     */
648
649          char vlarge_info_eyecatcher[8];
650          char *vlarge_buf_ptr;
651          uint32_t vlarge_buf_length;
652          uint32_t vlarge_current_use;
653          char aggrec_rsvdf[4];   /* pad to quadword boundary      */
654
655          /* ****** fsck log BUFFER ***** */
656
657  #define FSCKLOG_BUFSIZE (2 * BYTESPERPAGE)
658
659          char fscklog_info_eyecatcher[8];
660          struct fsck_blk_map_page *fscklog_buf_ptr;
661          uint32_t fscklog_buf_length;
662          uint32_t fscklog_buf_data_len;
663          int64_t fscklog_agg_offset;
664          int64_t fscklog_log_offset;
665          struct fscklog_entry_hdr *fscklog_last_msghdr;
666          int8_t fscklog_full;
667          int8_t fscklog_buf_allocated;
668          int8_t fscklog_buf_alloc_err;
669          char aggrec_rsvdg[5];   /* pad to quadword boundary        */
670
671          /* ****** fsck block map BUFFER *****
672           *
673           * N.B. If the fsck workspace block map is instantiated in
674           *      dynamic storage, then the following describe the entire
675           *      block map.  However, if the fsck workspace block map
676           *      is instantiated within the aggregate, then the following
677           *      describe the I/O buffer associated with it.
678           */
679
680  #define BLKMP_IO_BUFSIZE 4 * BYTESPERPAGE
681
682          char blkmp_info_eyecatcher[8];
683          struct fsck_blk_map_page *blkmp_buf_ptr;
684          uint32_t blkmp_buf_length;
685          uint32_t blkmp_buf_data_len;
686          int64_t blkmp_agg_offset;
687          int64_t blkmp_blkmp_offset;
688          int8_t blkmp_buf_write;
689          struct fsck_blk_map_hdr *blkmp_ctlptr;
690          int32_t blkmp_pagecount;
691
692          /* ****** BLOCK MAP DMAP PAGE I/O BUFFER ****** */
693          /*
694           * note: The EA buffer is treated differently from the
695           *       other I/O buffers allocated for fsck.
696           *       Specifically,
697           *
698           *            - it is larger than any other fsck I/O buffer
699           *              (sized to accomodate the largest legal ea)
700           *            - since it is exactly 1 memory segment, it is
701           *              not allocated as a normal wsp extent (described
702           *              by an fer on the wsp_extent_list) but is
703           *              allocated as a special case.
704           *            - this buffer is released as soon as phase 1 is
705           *              completed (does not persist until fsck has
706           *              finished processing)
707           *            - BTW the EA data is read by fsck, but never
708           *              written.
709           */
710
711  #define EA_IO_BUFSIZE   16*BYTESPERPAGE
712
713          char ea_info_eyecatcher[8];
714          char *ea_buf_ptr;
715          uint32_t ea_buf_length;
716          uint32_t ea_buf_data_len;
717          int64_t ea_agg_offset;
718
719          char aggrec_rsvdh[4];   /* pad to quadword boundary        */
720
```

```
721         /* ****** IAG I/O BUFFER ****** */
722         /*
723          * note: This is not actually a unique buffer.  Since
724          *       the Inode Allocation Maps verification is
725          *       completed before the Block Allocation Map
726          *       verification, buffer space is allocated and
727          *       used first for Inode Alloc Map verification
728          *       processing and then for Block Allocation Map
729          *       verification.
730          */
731
732 #define IAG_IO_BUFSIZE 4 * BYTESPERPAGE
733
734         char iag_info_eyecatcher[8];
735         char *iag_buf_ptr;
736         uint32_t iag_buf_length;
737         uint32_t iag_buf_data_len;
738         int64_t iag_agg_offset;
739         uint32_t iag_buf_1st_inode;     /* inode number of 1st in buffer   */
740         uint32_t iag_fsnum;
741         int8_t iag_for_aggregate;
742         int8_t iag_which_it;
743         int8_t iag_buf_write;
744         char aggrec_rsvdi[9];   /* pad to quadword boundary        */
745
746         /* ****** MAP CONTROL PAGE I/O BUFFER ****** */
747
748 #define MAPCTL_IO_BUFSIZE BYTESPERPAGE
749
750         char mapctl_info_eyecatcher[8];
751         char *mapctl_buf_ptr;
752         uint32_t mapctl_buf_length;
753         uint32_t mapctl_buf_data_len;
754         int64_t mapctl_agg_offset;
755         int8_t mapctl_buf_write;
756         char aggrec_rsvdk[3];   /* pad to quadword boundary        */
757
758         /* ****** MAP LEAF I/O BUFFER ****** */
759
760 #define MAPLEAF_IO_BUFSIZE   \
761               ((((XTPAGE_SIZE*4)+BYTESPERPAGE−1)/BYTESPERPAGE)*BYTESPERPAGE)
762
763         char maplf_info_eyecatcher[8];
764         char *mapleaf_buf_ptr;
765         uint32_t mapleaf_buf_length;
766         uint32_t mapleaf_buf_data_len;
767         int64_t mapleaf_agg_offset;
768         int8_t mapleaf_for_aggregate;
769         int8_t mapleaf_which_it;
770         int8_t mapleaf_buf_write;
771         char aggrec_rsvdm[1];   /* pad to quadword boundary        */
772
773         /* ****** BLOCK MAP LEVEL PAGE I/O BUFFER ****** */
774
775 #define BMAPLV_IO_BUFSIZE BYTESPERPAGE
776
777         char bmplv_info_eyecatcher[8];
778         char *bmaplv_buf_ptr;
779         uint32_t bmaplv_buf_length;
780         uint32_t bmaplv_buf_data_len;
781         int64_t bmaplv_agg_offset;
782         int64_t bmaplv_logical_offset;
783         int8_t bmaplv_current_level;
784         int8_t bmaplv_buf_write;
785         char aggrec_rsvdo[10];  /* pad to quadword boundary        */
786
787         /* ****** BLOCK MAP DMAP PAGE I/O BUFFER ****** */
788         /*
789          * note: This is not actually a unique buffer.  Since
790          *       the Inode Allocation Maps verification is
791          *       completed before the Block Allocation Map
792          *       verification, buffer space is allocated and
793          *       used first for Inode Alloc Map verification
794          *       processing and then for Block Allocation Map
795          *       verification.
796          */
797
798 #define BMAPDMP_IO_BUFSIZE IAG_IO_BUFSIZE
799
800         char bmpdm_info_eyecatcher[8];
801         char *bmapdm_buf_ptr;
802         uint32_t bmapdm_buf_length;
803         uint32_t bmapdm_buf_data_len;
804         int64_t bmapdm_agg_offset;
805         int64_t bmapdm_logical_offset;
806         int8_t bmapdm_buf_write;
807         char aggrec_rsvdq[11];  /* pad to quadword boundary        */
808
809         /* ****** INODE I/O BUFFER ****** */
810
```

```
811 #define INODE_IO_BUFSIZE  \
812             (((INODE_EXTENT_SIZE+BYTESPERPAGE−1)/BYTESPERPAGE)*BYTESPERPAGE)
813
814         char inobuf_info_eyecatcher[8];
815         char *ino_buf_ptr;
816         uint32_t ino_buf_length;
817         uint32_t ino_buf_data_len;
818         int64_t ino_buf_agg_offset;     /* agg byte offset of buf contents */
819         uint32_t ino_buf_1st_ino;       /* 1st inode number in buffer      */
820         uint32_t ino_fsnum;
821         pxd_t ino_ixpxd;
822         int8_t ino_for_aggregate;
823         int8_t ino_which_it;
824         int8_t ino_buf_write;
825         char aggrec_rsvds[1];   /* pad to quadword boundary        */
826
827         /* ****** INTERNAL/LEAF NODE I/O BUFFER ****** */
828
829 #define NODE_IO_BUFSIZE  \
830             ((((XTPAGE_SIZE*4)+BYTESPERPAGE−1)/BYTESPERPAGE)*BYTESPERPAGE)
831
832         char nodbuf_info_eyecatcher[8];
833         char *node_buf_ptr;
834         uint32_t node_buf_length;
835         uint32_t node_buf_data_len;
836         int64_t node_agg_offset;
837         int8_t node_buf_write;
838         char aggrec_rsvdu[3];   /* pad to quadword boundary        */
839
840         /* ************************** instantiation of imap AG tables */
841
842         char agg_AGTbl_eyecatcher[8];
843         char aggrec_rsvdw[8];   /* pad to quadword boundary    */
844         struct fsck_ag_record agg_AGTbl[MAXAG];
845
846         char fset_AGTbl_eyecatcher[8];
847         char aggrec_rsvdy[8];   /* pad to quadword boundary    */
848         struct fsck_ag_record fset_AGTbl[MAXAG];
849
850         /* ********************** instantiation of imap IAG workspace */
851         char amap_eyecatcher[8];
852         uint32_t amap[EXTSPERIAG];      /* 512 : inode allocation map   */
853         char aggrec_rsvd00[8];  /* pad to quadword boundary     */
854         char fextsumm_eyecatcher[8];
855         int32_t fextsumm[SMAPSZ];       /*  16 : free extent summary map */
856         char aggrec_rsvd02[8];  /* pad to quadword boundary     */
857         char finosumm_eyecatcher[8];
858         int32_t finosumm[SMAPSZ];       /*  16 : free inode summary map */
859
860         /* ********************** instantiation of blockmap workspace */
861
862         struct blkmap_wspace blkmap_wsp;
863 };
864
865 #endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_INODE
#define H_INODE

#include <jfs_types.h>
#include "devices.h"

int ujfs_rwinode(HFILE, struct dinode *, uint32_t, int32_t, int32_t, uint32_t, uint32_t);
int ujfs_rwdaddr(HFILE, int64_t *, struct dinode *, int64_t, int32_t, int32_t);

#endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_UJFS_ENDIAN
#define _H_UJFS_ENDIAN

#include "jfs_types.h"
#include "jfs_byteorder.h"
#include "jfs_superblock.h"
#include "jfs_dmap.h"
#include "jfs_imap.h"
#include "jfs_dinode.h"
#include "jfs_logmgr.h"
#include "fsckwsp.h"

#if __BYTE_ORDER == __BIG_ENDIAN
void ujfs_swap_dbmap(struct dbmap *);
void ujfs_swap_dinode(struct dinode *, int32_t, uint32_t);
void ujfs_swap_dinomap(struct dinomap *);
void ujfs_swap_dmap(struct dmap *);
void ujfs_swap_dmapctl(struct dmapctl *);
void ujfs_swap_dtpage_t(dtpage_t *, uint32_t);
void ujfs_swap_fsck_blk_map_hdr(struct fsck_blk_map_hdr *);
void ujfs_swap_fsck_blk_map_page(struct fsck_blk_map_page *);
void ujfs_swap_fscklog_entry_hdr(struct fscklog_entry_hdr *);
void ujfs_swap_iag(struct iag *);
void ujfs_swap_logpage(struct logpage *, uint8_t);
void ujfs_swap_logsuper(struct logsuper *);
void ujfs_swap_lrd(struct lrd *);
void ujfs_swap_superblock(struct superblock *);
void ujfs_swap_xtpage_t(xtpage_t *);

static inline void ujfs_swap_inoext(struct dinode *ptr, int32_t mode, uint32_t flag)
{
        int i;
        for (i = 0; i < INOSPEREXT; i++, ptr++)
                ujfs_swap_dinode(ptr, mode, flag);
}

#define swap_multiple(swap_func, ptr, num)      \
do {                                            \
        int i;                                  \
        for (i = 0; i < num; i++, (ptr)++)      \
                swap_func(ptr);                 \
} while (0)

#else                           /* Little endian */

#define ujfs_swap_dbmap(dbmap)                  do {} while (0)
#define ujfs_swap_dinode(dinode, mode, flag)    do {} while (0)
#define ujfs_swap_dinomap(dinomap)              do {} while (0)
#define ujfs_swap_dmap(dmap)                    do {} while (0)
#define ujfs_swap_dmapctl(dmapctl)              do {} while (0)
#define ujfs_swap_dtpage_t(dtpage, flag)        do {} while (0)
#define ujfs_swap_fsck_blk_map_hdr(map_hdr)     do {} while (0)
#define ujfs_swap_fsck_blk_map_page(map_page)   do {} while (0)
#define ujfs_swap_fscklog_entry_hdr(ent_hdr)    do {} while (0)
#define ujfs_swap_iag(iag)                      do {} while (0)
#define ujfs_swap_inoext(inoext, mode, flag)    do {} while (0)
#define ujfs_swap_logpage(logpage, pages)       do {} while (0)
#define ujfs_swap_logsuper(logsuper)            do {} while (0)
#define ujfs_swap_lrd(lrd)                      do {} while (0)
#define ujfs_swap_superblock(sb)                do {} while (0)
#define ujfs_swap_xtpage_t(xtpage)              do {} while (0)
#define swap_multiple(swap_func, ptr, num)      do {} while (0)

#endif

#endif                          /* _H_UJFS_ENDIAN */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_LIBFS
#define H_LIBFS

#define LIBFS_BADMAGIC          −5      /* magic number not recognized      */
#define LIBFS_BADVERSION        −6      /* magic num ok, incompatible vers  */
#define LIBFS_CORRUPTSUPER      −10     /* fragsize, agsize, or iagsize bad */

#endif                          /* H_LIBFS */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_LOGFORM
#define H_LOGFORM

int jfs_logform(int, int, int, uint, int64_t, int, uuid_t, char *);

#endif                          /* H_LORFORM */
```

```c
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "jfs_types.h"
#include "utilsubs.h"

/*
 *      log2shift()
 */
int32_t log2shift(uint32_t n)
{
        uint32_t shift = 0;

        while (n > 1) {
                /* n is not power of 2 */
                if (n & 1)
                        return −1;

                shift++;
                n >>= 1;
        }

        return shift;
}


/*
 *      ui
 *      ==
 */
/*
 *      prompt()
 */
char prompt(char *str)
{
        char cmd[81];

        fputs(str, stdout);
        fflush(stdout);

        /* get NULL terminated input */
        fgets(cmd, 81, stdin);

        return cmd[0];          /* return response letter */
}


/*
 *      more()
 */
int more(void)
{
        char cmd[81];

        fputs("− hit Enter to continue, e[x]it −", stdout);
        fflush(stdout);

        /* get NULL terminated input */
        fgets(cmd, 80, stdin);

        if (cmd[0] == 'x')
                return 1;       /* do NOT continue */
        else
                return 0;       /* continue */
}
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef _H_UJFS_UTILSUBS
#define _H_UJFS_UTILSUBS

/*
 *      utilsubs.h
 */

/*
 *      function prototypes
 */
int32_t log2shift(uint32_t n);
char prompt(char *str);
int more(void);

#endif                          /* _H_UJFS_UTILSUBS */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *   FUNCTIONS: no−frills substitutes for fsck routines
 *              used by logredo modules outside of fsck
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "jfs_types.h"
#include "fsckmsgc.h"
#include "fsckmsge.h"

#define STDOUT_HANDLE  1
#define STDERR_HANDLE  2

extern short MsgProtocol[][2];

extern char *Vol_Label;

extern char *verbose_msg_ptr;

extern char *msgprms[];
extern short msgprmidx[];

extern unsigned long msgs_txt_maxlen;

/*****************************************************************************
 * NAME: alloc_wrksp
 *
 * FUNCTION:  Allocates and initializes (to guarantee the storage is backed)
 *            dynamic storage for the caller.
 *
 * PARAMETERS:
 *      length         − input − the number of bytes of storage which are needed
 *      dynstg_object  − input − a constant (see xfsck.h) identifying the purpose
 *                               for which the storage is needed (Used in error
 *                               message if the request cannot be satisfied.
 *      addr_wrksp_ptr − input − the address of a variable in which this routine
 *                               will return the address of the dynamic storage
 *                               allocated for the caller
 *
 * NOTES:
 *
 * RETURNS:
 *      success: 0
 *      failure: something else
 */
int alloc_wrksp(unsigned length, int dynstg_object, int for_logredo, void **addr_wrksp_ptr)
{
        int awsp_rc = 0;
        unsigned min_length;

        *addr_wrksp_ptr = NULL; /* initialize return value */
        min_length = ((length + 7) / 4) * 4;    /* round up to an 4 byte boundary */

        *addr_wrksp_ptr = (char *) malloc(min_length);

        return (awsp_rc);
}                               /* end alloc_wrksp() */

/*****************************************************************************
 * NAME: fsck_send_msg
 *
 * FUNCTION: Issue an fsck message, depending on the message's protocol
 *           according to the fsck message arrays.
 *
 * PARAMETERS:
 *      ?                − input −
 *      ?                − returned −
 *
 * NOTES:  none
 *
 * RETURNS:
```

```
 * nothing
 */
void fsck_send_msg(int msg_num, int num_inserts)
{
        int prmidx;

        memset((void *) verbose_msg_ptr, '\0', msgs_txt_maxlen);

        prmidx = 0;
        while (prmidx < num_inserts) {
                if (msgprmidx[prmidx] != 0) {   /* variable text insert */
                        msgprms[prmidx] = MsgText[msgprmidx[prmidx]];
                }               /* end variable text insert */
                prmidx += 1;
        }

        sprintf(verbose_msg_ptr, MsgText[msg_num],
                msgprms[0], msgprms[1], msgprms[2], msgprms[3], msgprms[4],
                msgprms[5], msgprms[6], msgprms[7], msgprms[8], msgprms[9]);

        printf("%s", verbose_msg_ptr);

        return;
}                               /* end fsck_send_msg() */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_INITMAP
#define H_INITMAP

#define ALLOC           0x1
#define FREE            0x2
#define BADBLOCK        0x4

int calc_map_size(int64_t, struct dinode *, int, int *, unsigned);
int markit(int, unsigned);
int record_LVM_BadBlks( int, int, int, struct dinode *, int64_t );
int verify_last_blocks( int, int, struct dinode * );
int write_block_map(int, int64_t, int);

#endif  /* H_INITMAP */
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#ifndef H_INODEMAP
#define H_INODEMAP

#include "devices.h"

int init_inode_map(int, HFILE, int64_t, int, int64_t, int, unsigned short, int, unsigned);

#endif
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */

/* JFS includes */
#include "jfs_filsys.h"
#include "jfs_superblock.h"
#include "jfs_logmgr.h"

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * NAME: build_flag_string
 *
 * FUNCTION: build a string of text descriptions of
 *           the set flags in the superblock
 *
 * PARAMETERS:
 *      flags       − flags field from the superblock
 *      flag_string − returned string of flag descriptions
 */
void build_flag_string(uint32_t flags, char *flag_string)
{
        char *string_ptr = flag_string;

        if (flags & JFS_LINUX) {
                strcpy(string_ptr, "JFS_LINUX ");
                string_ptr += 11;
        }
        if (flags & JFS_OS2) {
                strcpy(string_ptr, "JFS_OS2 ");
                string_ptr += 9;
        }
        if (flags & JFS_COMMIT) {
                strcpy(string_ptr, "JFS_COMMIT ");
                string_ptr += 12;
        }
        if (flags & JFS_GROUPCOMMIT) {
                strcpy(string_ptr, "JFS_GROUPCOMMIT ");
                string_ptr += 17;
        }
        if (flags & JFS_LAZYCOMMIT) {
                strcpy(string_ptr, "JFS_LAZYCOMMIT ");
                string_ptr += 16;
        }
        if (flags & JFS_INLINELOG) {
                strcpy(string_ptr, "JFS_INLINELOG ");
                string_ptr += 15;
        }
        if (flags & JFS_BAD_SAIT) {
                strcpy(string_ptr, "JFS_BAD_SAIT ");
                string_ptr += 14;
        }
        if (flags & JFS_SPARSE) {
                strcpy(string_ptr, "JFS_SPARSE ");
                string_ptr += 12;
        }
        if (flags & JFS_DASD_ENABLED) {
                strcpy(string_ptr, "JFS_DASD_ENABLED ");
                string_ptr += 18;
        }
        if (flags & JFS_DASD_PRIME) {
                strcpy(string_ptr, "JFS_DASD_PRIME ");
                string_ptr += 16;
        }
        strcpy(string_ptr, "\0");

        return;
}

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * NAME: display_super
 *
 * FUNCTION: display the JFS file system superblock
 *
 * PARAMETERS:
 *      sb  − pointer to the file system superblock
 *
```

```
 * SAMPLE OUTPUT:

 JFS magic number:      'JFS1'
 JFS version:           2
 JFS state:             clean
 JFS flags:             JFS_LINUX  JFS_COMMIT  JFS_GROUPCOMMIT
 Aggregate block size:  4096 bytes
 Aggregate size:        2055784 blocks
 Physical block size:   512 bytes
 Allocation group size: 8192 aggregate blocks
 Log device number:     0x3f06
 Filesystem creation:   Fri Aug  2 14:48:53 2002
 File system UUID:      7940a3b8-0bb0-4c0d-a389-630619783c6e
 Volume label:          '              '
 External log UUID:     47c80a1d-7f25-4c61-9596-f33e463e5b38

 */
void display_super(struct superblock *sb)
{
        char *state;
        char uuid_unparsed[37];
        char flag_string[142];
        time_t tm;

        switch (sb->s_state) {
        case FM_CLEAN:
                state = "clean";
                break;
        case FM_MOUNT:
                state = "mounted";
                break;
        case FM_DIRTY:
                state = "dirty";
                break;
        case FM_LOGREDO:
                state = "logredo";
                break;
        default:
                state = "unknown";
                break;
        }

        build_flag_string(sb->s_flag, flag_string);

        printf("\nJFS filesystem superblock:\n\n");
        printf("JFS magic number:\t'%4.4s'\n", sb->s_magic);
        printf("JFS version:\t\t%d\n", sb->s_version);
        printf("JFS state:\t\t%s\n", state);
        printf("JFS flags:\t\t%s\n", flag_string);
        printf("Aggregate block size:\t%d bytes\n", sb->s_bsize);
        printf("Aggregate size:\t\t%lld blocks\n", (long long) sb->s_size);
        printf("Physical block size:\t%d bytes\n", sb->s_pbsize);
        printf("Allocation group size:\t%d aggregate blocks\n", sb->s_agsize);
        printf("Log device number:\t0x%x\n", sb->s_logdev);
        tm = sb->s_time.tv_sec;
        printf("Filesystem creation:\t%s", ctime(&tm));
        /*
         * JFS version 2 incorporated new fields for
         * UUID, log UUID, and a 16 char volume label
         * into the superblock.  Account for those here.
         */
        if (sb->s_version == JFS_VERSION) {
                uuid_unparse(sb->s_uuid, uuid_unparsed);
                printf("File system UUID:\t%s\n", uuid_unparsed);
                printf("Volume label:\t\t'%16s'\n", sb->s_label);
                uuid_unparse(sb->s_loguuid, uuid_unparsed);
                printf("External log UUID:\t%s\n", uuid_unparsed);
        } else {
                printf("Volume label:\t\t'%11s'\n", sb->s_fpack);
        }
        printf("\n");

        return;
}

/*--------------------------------------------------------------------
 * NAME: display_logsuper
 *
 * FUNCTION: display the JFS log superblock
 *
 * PARAMETERS:
 *      lsp  - pointer to the log superblock
 *
 * SAMPLE OUTPUT:

 JFS log magic number:  0x87654321
 JFS log version:       1
 Log opened/mounted:    3
 Logical block size:    4096 bytes
 Log size:              32768 blocks
```

```
 Log flags:             JFS_LINUX  JFS_COMMIT  JFS_GROUPCOMMIT
 Log state:             LOGMOUNT
 Log device UUID:       47c80a1d-7f25-4c61-9596-f33e463e5b38
 Log volume label:      '       JFSLogVol'
 Active file systems:
     active[0]:  7940a3b8-0bb0-4c0d-a389-630619783c6e

 */
void display_logsuper(struct logsuper * lsp)
{
        char *state;
        char uuid_unparsed[37];
        int i;
        bool active_fs = false;
        char flag_string[142];

        switch (lsp->state) {
        case LOGMOUNT:
                state = "LOGMOUNT";
                break;
        case LOGREDONE:
                state = "LOGREDONE";
                break;
        case LOGWRAP:
                state = "LOGWRAP";
                break;
        case LOGREADERR:
                state = "LOGREADERR";
                break;
        default:
                state = "Unknown";
                break;
        }

        build_flag_string(lsp->flag, flag_string);

        printf("\nJFS external log superblock:\n\n");
        printf("JFS log magic number:\t0x%x\n", lsp->magic);
        printf("JFS log version:\t%d\n", lsp->version);
        printf("Log opened/mounted:\t%d\n", lsp->serial);
        printf("Logical block size:\t%d bytes\n", lsp->bsize);
        printf("Log size:\t\t%d blocks\n", lsp->size);
        printf("Log flags:\t\t%s\n", flag_string);
        printf("Log state:\t\t%s\n", state);
        uuid_unparse(lsp->uuid, uuid_unparsed);
        printf("Log device UUID:\t%s\n", uuid_unparsed);
        printf("Log volume label:\t'%16s'\n", lsp->label);
        printf("Active file systems:");
        for (i = 0; i < MAX_ACTIVE; i++) {
                uuid_unparse(lsp->active[i], uuid_unparsed);
                /* only print active (non-zero uuid) file systems */
                if (strncmp(uuid_unparsed, "00000000-0000-0000-0000-000000000000", 36)) {
                        printf("\n  active[%d]: %s", i, uuid_unparsed);
                        active_fs = true;
                }
        }

        if (!active_fs) {
                printf("\tNone active.\n");
        }
        printf("\n");

        return;
}
```

```c
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#include "config.h"

#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <ctype.h>
#include "jfs_types.h"
#include "jfs_endian.h"
#include "jfs_filsys.h"
#include "jfs_superblock.h"
#include "inode.h"
#include "super.h"
#include "jfs_version.h"

#define EXIT(fd, rc)  {close(fd); exit(rc);}

static int J_flag, l_flag, L_flag, U_flag;
char *new_label, *new_UUID;
char *device;
char logdev[255] = { '\0' };
int log_fd = −1;

extern int LogOpenMode;
extern int open_by_label(uuid_t, int, int, char *);

void tune_usage()
{
        printf("\nUsage:  jfs_tune [−J options] [−l] [−L vol_label] [−U uuid] [−V] device\n"
               "\nEmergency help:\n"
               " −J options    Set external journal options.\n"
               " −l            Display superblock\n"
               " −L vol_label  Set volume label.\n"
               " −U uuid       Set UUID.\n"
               " −V            Print version information only.\n");
        exit(−1);
}

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * NAME: parse_journal_opts
 *
 * FUNCTION: parse journal (−J) options
 *           set log file descriptor (global log_fd)
 *           set log device name (global logdev)
 *
 * PARAMETERS:
 *      opts − options string
 */
void parse_journal_opts(const char *opts)
{
        int journal_usage = 0;
        uuid_t log_uuid;

        LogOpenMode = O_RDWR | O_EXCL;

        if (strncmp(opts, "device=", 7) == 0) {
                if (strncmp(opts + 7, "UUID=", 5) == 0) {
                        if (uuid_parse((char *) opts + 7 + 5, log_uuid)) {
                                fputs("\nError: UUID entered in improper format.\n",
                                      stderr);
                                exit(−1);
                        } else {
                                log_fd = open_by_label(log_uuid, 0, 1, logdev);
                        }
                } else if (strncmp(opts + 7, "LABEL=", 6) == 0) {
                        log_fd = open_by_label((char *) opts + 7 + 6, 1, 1, logdev);
                } else {
                        strcpy(logdev, ((char *) opts + 7));
                        if (logdev) {
                                log_fd = open(logdev, LogOpenMode, 0);
```

```
                        } else {
                                journal_usage++;
                        }
                }
        } else
                journal_usage++;

        if (journal_usage) {
                fprintf(stderr, "\nInvalid journal option(s) specified.\n\n"
                                "Valid options for −J are:\n"
                                "\tdevice=<journal device>\n"
                                "\tdevice=UUID=<UUID of journal device>\n"
                                "\tdevice=LABEL=<label of journal device>\n\n");
                exit(1);
        }

        return;
}

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * NAME: parse_tune_options
 *
 * FUNCTION: parse tune options
 *
 * PARAMETERS:
 *      argc − number of passed arguments
 *      argv − string of arguments
 *
 * RETURNS:
 *      success: 0
 *      failure: any other value
 */
static void parse_tune_options(int argc, char *argv[])
{
        int c;

        while ((c = getopt(argc, argv, "J:lL:U:V")) != EOF) {
                switch (c) {
                case 'J':
                        /* attach external journal device */
                        parse_journal_opts(optarg);
                        J_flag = 1;
                        break;
                case 'l':
                        /* display superblock */
                        l_flag = 1;
                        break;
                case 'L':
                        /* set volume label */
                        new_label = optarg;
                        L_flag = 1;
                        break;
                case 'U':
                        /* set UUID */
                        new_UUID = optarg;
                        U_flag = 1;
                        break;
                case 'V':
                        /* print version and exit */
                        exit(0);
                        break;
                default:
                        tune_usage();
                        break;
                }
        }

        if (optind != argc − 1) {
                printf("\nError: Device not specified or command format error.\n");
                tune_usage();
        }

        if (!J_flag && !l_flag && !L_flag && !U_flag) {
                printf("\nError: No options selected.\n");
                tune_usage();
        }

        device = argv[optind];

        return;
}

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
 * NAME: main
 *
 * FUNCTION: adjust JFS tunable parameters
 *
 * PARAMETERS:
 *      argc − number of passed arguments
 *      argv − string of arguments
```

```
 *
 * RETURNS:
 *      success: 0
 *      failure: any other value
 */
int main(int argc, char *argv[])
{
        char *argp;
        int rc = 0;
        int fd, superblock_type;
        bool mounted = false;
        struct superblock sb;
        struct logsuper logsup;

#define FS_SUPER_SECONDARY  0
#define FS_SUPER_PRIMARY    1
#define LOG_SUPER           2

        printf("jfs_tune version %s, %s\n", VERSION, JFSUTILS_DATE);

        parse_tune_options(argc, argv);

        /*
         * Check if device is mounted.  -l is the only parameter
         * supported on a mounted device, so if any others are
         * selected, let the user know.  If the device is mounted
         * and -l was not specified, get out.
         */
        if (rc = Is_Device_Mounted(device)) {
                mounted = true;
                if (J_flag || L_flag || U_flag) {
                        fprintf(stderr, "\n%s is mounted.\n"
                                "While mounted, the only supported jfs_tune parameter is -l.\n",
                                device);
                        if (!l_flag) {
                                exit(-1);
                        }
                }
        }

        /* Open device */
        fd = open(device, O_RDWR | O_EXCL, 0);
        if (fd < 0) {
                fprintf(stderr, "Error: Cannot open device %s.\n", device);
                exit(-1);
        }

        /* Get and validate primary JFS superblock */
        if ((rc = ujfs_get_superblk(fd, &sb, 1)) == 0) {
                if ((rc = ujfs_validate_super(&sb)) == 0) {
                        superblock_type = FS_SUPER_PRIMARY;
                }
        }

        /* If failure retrieving primary superblock, get/validate secondary superblock */
        if (rc) {
                if ((rc = ujfs_get_superblk(fd, &sb, 0)) == 0) {
                        if ((rc = ujfs_validate_super(&sb)) == 0) {
                                superblock_type = FS_SUPER_SECONDARY;
                        }
                }
        }

        /* If no valid FS superblock, see if we have a log superblock */
        if (rc) {
                if ((rc = ujfs_get_logsuper(fd, &logsup)) == 0) {
                        if ((rc = ujfs_validate_logsuper(&logsup)) == 0) {
                                superblock_type = LOG_SUPER;
                                /*
                                 * We know this is an external journal device.
                                 * Now check to see if it is attached to a
                                 * mounted file system.  If so, the only
                                 * valid option is -l.
                                 */
                                if (logsup.state == LOGMOUNT) {
                                        mounted = true;
                                        if (J_flag || L_flag || U_flag) {
                                                fprintf(stderr,
                                                        "\n%s contains an external journal for a mounted filesystem.\n"
                                                        "While mounted, the only supported jfs_tune parameter is -l.\n",
                                                        device);
                                                if (!l_flag) {
                                                        EXIT(fd, -1);
                                                }
                                        }
                                }
                        }
                }
        }

```

```
271         /* If we couldn't find/read a valid JFS FS or log superblock, warn user and exit */
272         if (rc) {
273                 printf("\nCould not read valid JFS FS or log superblock on device %s.\n",
274                        device);
275                 EXIT(fd, -1);
276         }
277
278         /*
279          * Account for bug in mkfs.jfs 1.0.18 and 1.0.19
280          * that didn't properly set the file system version
281          * number to 2 when using an external journal.
282          */
283         if (!mounted && (superblock_type < LOG_SUPER) &&
284             (sb.s_version < JFS_VERSION) && !(sb.s_flag & JFS_INLINELOG)) {
285                 sb.s_version = JFS_VERSION;
286                 rc = ujfs_put_superblk(fd, &sb, superblock_type);
287                 if (rc) {
288                         printf("\nCould not update JFS version number properly on %s.\n",
289                                device);
290                         EXIT(fd, rc);
291                 }
292         }
293
294         /*
295          * set volume label on unmounted device
296          */
297         if (L_flag && !mounted) {
298                 if (superblock_type < LOG_SUPER) {
299                         /* change label in JFS file system superblock */
300                         /*
301                          * The superblock in JFS releases before 1.0.18 stores
302                          * the label in s_fpack[11].  The superblock in JFS
303                          * releases 1.0.18 and greater has s_fpack, but uses
304                          * the new field s_label[16] to store the label.
305                          * s_label is in an area of the superblock that was
306                          * allocated but unused in pre 1.0.18, so if per chance
307                          * the user is using an old JFS file system, setting
308                          * s_label will not be a problem.
309                          */
310                         memset(sb.s_fpack, 0, sizeof (sb.s_fpack));
311                         strncpy(sb.s_fpack, new_label, sizeof (sb.s_fpack));
312                         if (strlen(new_label) > sizeof (sb.s_label))
313                                 fprintf(stderr, "Warning: label too long, truncating.\n");
314                         memset(sb.s_label, 0, sizeof (sb.s_label));
315                         strncpy(sb.s_label, new_label, sizeof (sb.s_label));
316                         rc = ujfs_put_superblk(fd, &sb, superblock_type);
317                 } else {
318                         /* change label in JFS log superblock */
319                         if (strlen(new_label) > sizeof (logsup.label))
320                                 fprintf(stderr, "Warning: label too long, truncating.\n");
321                         memset(logsup.label, 0, sizeof (logsup.label));
322                         strncpy(logsup.label, new_label, sizeof (logsup.label));
323                         rc = ujfs_put_logsuper(fd, &logsup);
324                 }
325                 if (rc) {
326                         printf("\nError writing superblock to disk.  Label unchanged.\n");
327                         EXIT(fd, rc);
328                 } else {
329                         printf("Volume label updated successfully.\n");
330                 }
331         }
332
333         /*
334          * set UUID on umounted device
335          */
336         if (U_flag && !mounted) {
337                 uuid_t *uu;
338                 uu = ((superblock_type < LOG_SUPER) ? &sb.s_uuid : &logsup.uuid);
339                 if ((strcasecmp(new_UUID, "null") == 0) ||
340                     (strcasecmp(new_UUID, "clear") == 0)) {
341                         uuid_clear(*uu);
342                 } else if (strcasecmp(new_UUID, "time") == 0) {
343                         uuid_generate_time(*uu);
344                 } else if (strcasecmp(new_UUID, "random") == 0) {
345                         uuid_generate(*uu);
346                 } else if (uuid_parse(new_UUID, *uu)) {
347                         fprintf(stderr, "Invalid UUID format.\n");
348                         EXIT(fd, -1);
349                 }
350
351                 if (superblock_type < LOG_SUPER) {
352                         rc = ujfs_put_superblk(fd, &sb, superblock_type);
353                 } else {
354                         rc = ujfs_put_logsuper(fd, &logsup);
355                 }
356
357                 if (rc) {
358                         printf("\nError writing superblock to disk.  UUID unchanged.\n");
359                         EXIT(fd, rc);
360                 } else {
```

```
361                             printf("UUID updated successfully.\n");
362                     }
363             }
364
365             /*
366              * attach external journal to JFS file system
367              */
368             if (J_flag && !mounted) {
369                     /*
370                      * NOTE: If we ever allow attaching more than one file system to a
371                      * single log file, we'll have to change the conditions of the above
372                      * 'if' to account for a log file that is in use by one file system
373                      * (state LOGMOUNT), but is being attached to by another file system.
374                      */
375                     struct stat st;
376
377                     /* make sure device to be attached to is a JFS file system */
378                     if (superblock_type >= LOG_SUPER) {
379                             printf("\nError: %s does not contain a JFS file system.\n",device);
380                             EXIT(fd, -1);
381                     }
382
383                     /* log_fd was set in parse_journal_opts */
384                     if (log_fd < 0) {
385                             printf("\nError: Could not find/open specified external journal device.\n");
386                             EXIT(fd, -1);
387                     }
388
389                     /* get valid log superblock */
390                     if ((rc = ujfs_get_logsuper(log_fd, &logsup)) == 0) {
391                             if ((rc = ujfs_validate_logsuper(&logsup)) == 0) {
392                                     if (fstat(log_fd, &st)) {
393                                             rc = -1;
394                                     } else {
395                                             /* update FS superblock */
396                                             sb.s_logdev = dev_to_kdev(st.st_rdev);
397                                             uuid_copy(sb.s_loguuid, logsup.uuid);
398                                             sb.s_version = JFS_VERSION;
399                                             sb.s_flag &= (~JFS_INLINELOG);
400                                             memset(&sb.s_logpxd, 0, sizeof (pxd_t));
401                                             rc = ujfs_put_superblk(fd, &sb, superblock_type);
402                                     }
403                             }
404                     }
405
406                     /* If we could't find/read a valid JFS log superblock, let user know */
407                     if (rc) {
408                             printf("\nError attaching JFS external journal to JFS FS.\n");
409                     } else {
410                             printf("Attached JFS external journal to JFS FS successfully.\n");
411                     }
412
413                     close(log_fd);
414             }
415
416             /*
417              * display superblock
418              */
419             if (l_flag) {
420                     if (superblock_type < LOG_SUPER) {
421                             display_super(&sb);
422                     } else {
423                             display_logsuper(&logsup);
424                     }
425             }
426
427             close(fd);
428             return rc;
429     }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
#include "xpeek.h"

/* libfs includes */
#include "devices.h"
#include "utilsubs.h"

void alter()
{
        int64_t address;
        int64_t block;
        uint8_t *buffer;
        uint8_t byte;
        char cmdline[512];
        char *hexstring;
        unsigned hex_length;
        unsigned length;
        unsigned offset;
        char *ptr;
        char *token;

        token = strtok(0, "     \n");
        if (token == 0) {
                fputs("alter: Please enter: block offset hex−digits> ", stdout);
                fgets(cmdline, 512, stdin);
                token = strtok(cmdline, "       \n");
                if (token == 0)
                        return;
        }
        errno = 0;
        block = strtoull(token, 0, 0);
        if (block == 0 && errno) {
                fputs("alter: invalid block\n\n", stderr);
                return;
        }
        address = block << l2bsize;

        token = strtok(0, "     \n");
        if (!token) {
                fputs("alter:  Not enought arguments!\n", stderr);
                return;
        }
        offset = strtoul(token, 0, 16);
        if (offset == 0 && errno) {
                fputs("alter: invalid offset\n", stderr);
                return;
        }
        hexstring = strtok(0, "         \n");
        if (!hexstring) {
                fputs("alter: Not enough arguments!\n", stderr);
                return;
        }
        if (strtok(0, "         \n")) {
                fputs("alter: Too many arguments!\n", stderr);
                return;
        }
        hex_length = strlen(hexstring);
        if (hex_length & 1) {
                /* odd number of hex digits */
                fputs("alter: hex string must have even number of digits!\n", stderr);
                return;
        }

        /* Round length of data up to next full physical block */

        length = (offset + hex_length + bsize − 1) & ~(bsize − 1);

        buffer = (char *) malloc(length);
        if (!buffer) {
                fputs("alter: malloc failure!\n", stderr);
                return;
        }
        if (ujfs_rw_diskblocks(fd, address, length, buffer, GET)) {
                fputs("alter: failed reading disk data!\n", stderr);
```

```
91                      free(buffer);
92                      return;
93              }
94              for (ptr = hexstring; *ptr; ptr++) {
95                      if (*ptr >= '0' && *ptr <= '9')
96                              byte = *ptr - '0';
97                      else if (*ptr >= 'a' && *ptr <= 'f')
98                              byte = *ptr - 'a' + 10;
99                      else if (*ptr >= 'A' && *ptr <= 'F')
100                             byte = *ptr - 'A' + 10;
101                     else {
102                             fputs("alter: invalid hex digit!\n", stderr);
103                             free(buffer);
104                             return;
105                     }
106
107                     ptr++;
108                     byte <<= 4;
109
110                     if (*ptr >= '0' && *ptr <= '9')
111                             byte += *ptr - '0';
112                     else if (*ptr >= 'a' && *ptr <= 'f')
113                             byte += *ptr - 'a' + 10;
114                     else if (*ptr >= 'A' && *ptr <= 'F')
115                             byte += *ptr - 'A' + 10;
116                     else {
117                             fputs("alter: invalid hex digit!\n", stderr);
118                             free(buffer);
119                             return;
120                     }
121
122                     buffer[offset++] = byte;
123             }
124             if (ujfs_rw_diskblocks(fd, address, length, buffer, PUT))
125                     fputs("alter: failed writing disk data!\n", stderr);
126
127             free(buffer);
128             return;
129     }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *   FUNCTION: Displays data in a variety of formats
 */
#include "xpeek.h"
#include "jfs_endian.h"

#include <jfs_xtree.h>

/* libfs includes */
#include <devices.h>

static void display_hex(char *, unsigned, unsigned);

extern unsigned type_jfs;

void display()
{
        int64_t addr;
        int64_t block;
        char *buffer;
        char cmdline[512];
        unsigned data_size;
        int format = 'a';
        int64_t len;
        unsigned length = 0;
        unsigned offset = 0;
        char *token;

        token = strtok(0, "     \n");
        if (token == 0) {
                fputs("display: Please enter: block [offset [format [count]]]\ndisplay> ", stdout);
                fgets(cmdline, 512, stdin);
                token = strtok(cmdline, "       \n");
                if (token == 0)
                        return;
        }
        errno = 0;
        block = strtoull(token, 0, 0);
        if (block == 0 && errno) {
                fputs("display: invalid block\n\n", stderr);
                return;
        }
        if ((token = strtok(0, "        \n"))) {
                offset = strtoul(token, 0, 16);
                if (offset == 0 && errno) {
                        fputs("display: invalid offset\n\n", stderr);
                        return;
                }
        }
        if ((token = strtok(0, "        \n")))
                format = token[0];

        if ((token = strtok(0, "        \n"))) {
                length = strtoul(token, 0, 0);
                if (length == 0 && errno) {
                        fputs("display: invalid length\n\n", stderr);
                        return;
                }
        }

        if (strtok(0, "         \n")) {
                fputs("display: Too many arguments\n\n", stderr);
                return;
        }

        switch (format) {
        case 'a':
                data_size = 1;
                if (length == 0)
                        length = bsize;
                break;
        case 'i':
                data_size = sizeof (struct dinode);
```

```
91                      if (length == 0)
92                              length = 1;
93                      break;
94              case 'I':
95                      data_size = sizeof (struct iag);
96                      if (length == 0)
97                              length = 1;
98                      break;
99              case 's':
100                     data_size = sizeof (struct superblock);
101                     if (length == 0)
102                             length = 1;
103                     break;
104             case 'x':
105                     data_size = 4;
106                     if (length == 0)
107                             length = bsize / 4;
108                     break;
109             case 'X':
110                     data_size = sizeof (xad_t);
111                     if (length == 0)
112                             length = 1;
113                     break;
114             default:
115                     fputs("display:  invalid format\n\n", stderr);
116                     return;
117             }
118
119             addr = block << l2bsize;
120             len = ((length * data_size) + offset + bsize − 1) & (~(bsize − 1));
121             buffer = malloc(len);
122             if (buffer == 0) {
123                     fputs("display: error calling malloc\n\n", stderr);
124                     return;
125             }
126
127             if (ujfs_rw_diskblocks(fd, addr, len, buffer, GET)) {
128                     fputs("display: ujfs_rw_diskblocks failed\n\n", stderr);
129                     free(buffer);
130                     return;
131             }
132
133             printf("Block: %lld   Real Address 0x%llx\n", (long long) block, (long long) addr);
134             switch (format) {
135             case 'a':
136             case 'x':
137                     display_hex(&buffer[offset], length, offset);
138                     break;
139             case 'i':
140                     {
141                             int i;
142                             struct dinode *inode = (struct dinode *) &buffer[offset];
143                             for (i = 0; i < length; i++, inode++) {
144                                     /* swap if on big endian machine */
145                                     ujfs_swap_dinode(inode, GET, type_jfs);
146                                     display_inode(inode);
147                                     if (more())
148                                             return;
149                             }
150                     }
151                     break;
152             case 'I':
153                     {
154                             int i;
155                             struct iag *iag = (struct iag *) & buffer[offset];
156                             for (i = 0; i < length; i++, iag++) {
157                                     /* swap if on big endian machine */
158                                     ujfs_swap_iag(iag);
159                                     display_iag(iag);
160                                     if (more())
161                                             return;
162                             }
163                     }
164                     break;
165
166             case 's':
167                     /* swap if on big endian machine */
168                     ujfs_swap_superblock((struct superblock *) &buffer[offset]);
169                     if (display_super((struct superblock *) &buffer[offset]) == XPEEK_CHANGED) {
170                             /* swap if on big endian machine */
171                             ujfs_swap_superblock((struct superblock *) &buffer[offset]);
172                             if (ujfs_rw_diskblocks(fd, addr, len, buffer, PUT))
173                                     fputs("Display:  Error writing superblock!\n", stderr);
174                     }
175
176                     break;
177             default:
178                     fputs("display:  specified format not yet supported\n\n", stderr);
179                     break;
180             }
```

```
181
182         free(buffer);
183         return;
184 }
185
186 /*
187  *      display_hex: display region in hex/ascii
188  */
189 static void display_hex(char *addr, unsigned length, unsigned offset)
190 {
191         uint8_t hextext[37];
192         uint8_t asciitxt[17];
193         uint8_t *x = (uint8_t *) addr, x1, x2;
194         int i, j, k, l;
195
196         hextext[36] = '\0';
197         asciitxt[16] = '\0';     /* null end of string */
198
199         l = 0;
200
201         for (i = 1; i <= ((length + 15) / 16); i++) {
202                 if (i > 1 && ((i - 1) % 16) == 0)
203                         if (more())
204                                 break;
205
206                 /* print address/offset */
207                 printf("%08x:", offset + l);
208
209                 /* print 16 bytes per line */
210                 for (j = 0, k = 0; j < 16; j++, x++, l++) {
211                         if ((j % 4) == 0)
212                                 hextext[k++] = ' ';
213                         if (l < length) {
214                                 hextext[k++] = ((x1 = ((*x & 0xf0) >> 4)) < 10)
215                                     ? ('0' + x1) : ('A' + x1 - 10);
216                                 hextext[k++] = ((x2 = (*x & 0x0f)) < 10)
217                                     ? ('0' + x2) : ('A' + x2 - 10);
218                                 asciitxt[j] = ((*x < 0x20) || (*x >= 0x7f)) ? '.' : *x;
219                         } else {        /* byte not in range */
220                                 hextext[k++] = ' ';
221                                 hextext[k++] = ' ';
222                                 asciitxt[j] = '.';
223                         }
224                 }
225                 printf("%s |%s|\n", hextext, asciitxt);
226         }
227 }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *       dmap − display/modify disk map
 */
#include "xpeek.h"

/* JFS includes */
#include <jfs_filsys.h>
#include <jfs_dmap.h>

/* libfs includes */
#include <jfs_endian.h>
#include <inode.h>
#include <devices.h>

/* ACTIONS */
#define DMAP_EXIT       0
#define DISPLAY_DBMAP   1
#define DISPLAY_CPAGE   2
#define DISPLAY_LEAF    3

/* DMAP BLOCK TYPES     */
#define DMAP    −1
#define LEVEL0  0
#define LEVEL1  1
#define LEVEL2  2

/* DMAP BLOCK CALCULATIONS */
#define L0FACTOR        (LPERCTL + 1)
#define L1FACTOR        ((L0FACTOR * LPERCTL) + 1)
#define L1PAGE(l1)      (2 + ((l1) * L1FACTOR))
#define L0PAGE(l1, l0)  (L1PAGE(l1) + 1 + (l0 * L0FACTOR))
#define DMAPPAGE(l1, l0, d)     (L0PAGE(l1, l0) + 1 + d)

/* forward references */
int decode_pagenum(int64_t page, int *l1, int *l0, int *dmap);
int display_agfree(int64_t * agfree);
int display_cpage(struct dinode *, int64_t *);
int display_dbmap(struct dbmap *, int64_t *, int64_t);
int display_leaf(struct dinode *, int64_t *);
int display_tree(struct dmapctl *, int64_t *, int *);

/* external references */
int display_map(unsigned *, int);       /* iag.c */

/* Global Data */
int dmap_level;                 /* Maximum level of dtree */
int dmap_l2bpp;                 /* log 2 of blocks per page */

extern unsigned type_jfs;

void dmap()
{
        int action;
        int64_t address;
        struct dinode bmap_inode;
        int64_t cntl_addr;
        struct dbmap cntl_page;
        int64_t lblock;         /* Logical block of BMAP to be displayed */

        lblock = 0;
        action = DISPLAY_DBMAP;

        /* Read block allocation map Inode */
        if (find_inode(BMAP_I, AGGREGATE_I, &address) ||
            xRead(address, sizeof (struct dinode), (char *) &bmap_inode)) {
                fputs("dmap: Error reading block allocation map inode\n", stderr);
                return;
        }

        /* swap if on big endian machine */
        ujfs_swap_dinode(&bmap_inode, GET, type_jfs);

        /* Read overall control page for the map */
```

```
91
92          if (ujfs_rwdaddr(fd, &cntl_addr, &bmap_inode, (int64_t) 0, GET, bsize) ||
93              xRead(cntl_addr, sizeof (struct dbmap), (char *) &cntl_page)) {
94                  fputs("dmap: Error reading aggregate dmap control page\n", stderr);
95                  return;
96          }
97
98          /* swap if on big endian machine */
99          ujfs_swap_dbmap(&cntl_page);
100
101         /* Useful stuff */
102         dmap_level = BMAPSZTOLEV(cntl_page.dn_mapsize);
103         dmap_l2bpp = cntl_page.dn_l2nbperpage;
104
105         while (action != DMAP_EXIT) {
106                 switch (action) {
107                 case DISPLAY_DBMAP:
108                         action = display_dbmap(&cntl_page, &lblock, cntl_addr);
109                         break;
110                 case DISPLAY_CPAGE:
111                         action = display_cpage(&bmap_inode, &lblock);
112                         break;
113                 case DISPLAY_LEAF:
114                         action = display_leaf(&bmap_inode, &lblock);
115                         break;
116                 default:
117                         fputs("dmap: Internal Error!\n", stderr);
118                         return;
119                 }
120         }
121         return;
122 }
123
124 /*****************************************************************************
125  **********************   Sample output from display_dbmap()
126
127 Block allocation map control page at block 128
128
129 [1] dn_mapsize:         0x00000031fa0   [9] dn_agheigth:        0
130 [2] dn_nfree:           0x00000030ed8   [10] dn_agwidth:        1
131 [3] dn_l2nbperpage:     3               [11] dn_agstart:        341
132 [4] dn_numag:           25              [12] dn_agl2size:       13
133 [5] dn_maxlevel:        0               [13] dn_agfree:         type 'f'
134 [6] dn_maxag:           16              [14] dn_agsize:         8192
135 [7] dn_agpref:          16              [15] pad:               Not Displayed
136 [8] dn_aglevel:         0
137
138 *****************************************************************************/
139
140 int display_dbmap(struct dbmap *hdr, int64_t *lblock, int64_t hdr_addr)
141 {
142         char cmdline[80];
143         int field;
144         int rc;
145         char *token;
146
147       dbmap_redisplay:
148         printf("\nBlock allocation map control page at block %lld\n\n",
149                 (long long) (hdr_addr >> l2bsize));
150         printf("[1] dn_mapsize:\t\t0x%011llx\t", (long long) hdr->dn_mapsize);
151         printf("[9] dn_agheigth:\t%d\n", hdr->dn_agheigth);
152         printf("[2] dn_nfree:\t\t0x%011llx\t", (long long) hdr->dn_nfree);
153         printf("[10] dn_agwidth:\t%d\n", hdr->dn_agwidth);
154         printf("[3] dn_l2nbperpage:\t%d\t\t", hdr->dn_l2nbperpage);
155         printf("[11] dn_agstart:\t%d\n", hdr->dn_agstart);
156         printf("[4] dn_numag:\t\t%d\t\t", hdr->dn_numag);
157         printf("[12] dn_agl2size:\t%d\n", hdr->dn_agl2size);
158         printf("[5] dn_maxlevel:\t%d\t\t", hdr->dn_maxlevel);
159         printf("[13] dn_agfree:\t\ttype 'f'\n");
160         printf("[6] dn_maxag:\t\t%d\t\t", hdr->dn_maxag);
161         printf("[14] dn_agsize:\t\t%lld\n", (long long) hdr->dn_agsize);
162         printf("[7] dn_agpref:\t\t%d\t\t", hdr->dn_agpref);
163         printf("[15] pad:\t\tNot Displayed\n");
164         printf("[8] dn_aglevel:\t\t%d\n", hdr->dn_aglevel);
165
166       dbmap_retry:
167         fputs("display_dbmap: [m]odify, [f]ree count, [t]ree, e[x]it > ", stdout);
168         fgets(cmdline, 80, stdin);
169         token = strtok(cmdline, "       \n");
170         if ((token == 0) || (token[0] == 't')) {
171                 switch (dmap_level) {
172                 case 2:
173                         *lblock = 1;    /* L2 page */
174                         break;
175                 case 1:
176                         *lblock = 2;    /* L1 page */
177                         break;
178                 case 0:
179                         *lblock = 3;    /* L0 page */
180                         break;
```

```
181                     default:
182                             fprintf(stderr, "display_dbmap:  dmap_level = %d\n", dmap_level);
183                             return DMAP_EXIT;
184                     }
185                     return DISPLAY_CPAGE;
186             }
187             if (token[0] == 'f') {
188                     rc = display_agfree(hdr->dn_agfree);
189
190                     /* swap if on big endian machine */
191                     ujfs_swap_dbmap(hdr);
192
193                     if ((rc & XPEEK_CHANGED) && xWrite(hdr_addr, sizeof (struct dbmap), (char *) hdr)) {
194                             fputs("display_dbmap: error writing control header\n\n", stderr);
195                             /* swap if on big endian machine */
196                             ujfs_swap_dbmap(hdr);
197                             return DMAP_EXIT;
198                     }
199
200                     /* swap if on big endian machine */
201                     ujfs_swap_dbmap(hdr);
202
203                     if (rc & XPEEK_REDISPLAY)
204                             goto dbmap_redisplay;
205
206                     return DMAP_EXIT;
207             }
208             if (token[0] != 'm') {  /* assume 'x' */
209                     return DMAP_EXIT;
210             }
211
212             field = m_parse(cmdline, 14, &token);
213             if (field == 0)
214                     goto dbmap_retry;
215
216             switch (field) {
217             case 1:
218                     hdr->dn_mapsize = strtoull(token, 0, 16);
219                     break;
220             case 2:
221                     hdr->dn_nfree = strtoull(token, 0, 16);
222                     break;
223             case 3:
224                     hdr->dn_l2nbperpage = strtoul(token, 0, 0);
225                     break;
226             case 4:
227                     hdr->dn_numag = strtoul(token, 0, 0);
228                     break;
229             case 5:
230                     hdr->dn_maxlevel = strtoul(token, 0, 0);
231                     break;
232             case 6:
233                     hdr->dn_maxag = strtoul(token, 0, 0);
234                     break;
235             case 7:
236                     hdr->dn_agpref = strtoul(token, 0, 0);
237                     break;
238             case 8:
239                     hdr->dn_aglevel = strtoul(token, 0, 0);
240                     break;
241             case 9:
242                     hdr->dn_agheigth = strtoul(token, 0, 0);
243                     break;
244             case 10:
245                     hdr->dn_agwidth = strtoul(token, 0, 0);
246                     break;
247             case 11:
248                     hdr->dn_agstart = strtoul(token, 0, 0);
249                     break;
250             case 12:
251                     hdr->dn_agl2size = strtoul(token, 0, 0);
252                     break;
253             case 13:
254                     fputs("display_dbmap: Can't change this field from here.\n", stderr);
255                     goto dbmap_retry;
256             case 14:
257                     hdr->dn_agsize = strtoull(token, 0, 0);
258                     break;
259             }
260
261             /* swap if on big endian machine */
262             ujfs_swap_dbmap(hdr);
263
264             if (xWrite(hdr_addr, sizeof (struct dbmap), (char *) hdr)) {
265                     fputs("display_dbmap: error writing control header\n\n", stderr);
266                     /* swap back if on big endian machine */
267                     ujfs_swap_dbmap(hdr);
268                     return DMAP_EXIT;
269             }
270
```

```
271         /* swap back if on big endian machine */
272         ujfs_swap_dbmap(hdr);
273
274         goto dbmap_redisplay;
275 }
276
277 /*****************************************************************************
278  **********************   Example Output of display_cpage()
279
280 Level 0 Control Page at block 136
281
282 [1] nleafs:    1024                    [5] budmin:     13
283 [2] l2nleafs:  10                      [6] stree:      hit <enter>
284 [3] leafidx:   341                     [7] pad:        Not Displayed
285 [4] height:    5
286
287 *****************************************************************************/
288 int display_cpage(struct dinode * bmap_inode, int64_t * lblock)
289 {
290         int64_t address;
291         int changed = 0;
292         char cmdline[80];
293         struct dmapctl ctl_page;
294         int field;
295         int rc;
296         char *token;
297         int dmap, l0, l1, type;
298
299         type = decode_pagenum(*lblock, &l1, &l0, &dmap);
300
301         if (ujfs_rwdaddr(fd, &address, bmap_inode, (*lblock) << dmap_l2bpp,
302                          GET, bsize) || xRead(address, sizeof (struct dmapctl), (char *) &ctl_page)) {
303                 fputs("display_cpage: Error reading control page!\n", stderr);
304                 return DMAP_EXIT;
305         }
306
307         /* swap if on big endian machine */
308         ujfs_swap_dmapctl(&ctl_page);
309
310       cpage_redisplay:
311         printf("\nLevel %d Control Page at block %lld\n\n", type, (long long) (address >> l2bsize));
312
313         printf("[1] nleafs:\t%d\t\t\t", ctl_page.nleafs);
314         printf("[5] budmin:\t%d\n", ctl_page.budmin);
315         printf("[2] l2nleafs:\t%d\t\t\t", ctl_page.l2nleafs);
316         printf("[6] stree:\thit <enter>\n");
317         printf("[3] leafidx:\t%d\t\t\t", ctl_page.leafidx);
318         printf("[7] pad:\tNot Displayed\n");
319         printf("[4] height:\t%d\n", ctl_page.height);
320
321       cpage_retry:
322         fputs("[m]odify, [u]p, [r]ight or [l]eft sibling, e[x]it, [s]tree > ", stdout);
323
324         fgets(cmdline, 80, stdin);
325         token = strtok(cmdline, "       \n");
326         if ((token == 0) || (token[0] == 's')) {
327                 rc = display_tree(&ctl_page, lblock, &changed);
328
329                 /* swap if on big endian machine */
330                 ujfs_swap_dmapctl(&ctl_page);
331
332                 if (changed && xWrite(address, sizeof (struct dmapctl), (char *) &ctl_page)) {
333                         fputs("display_cpage: Error writing control page!\n", stderr);
334                         /* swap back if on big endian machine */
335                         ujfs_swap_dmapctl(&ctl_page);
336                         return DMAP_EXIT;
337                 }
338
339                 /* swap if on big endian machine */
340                 ujfs_swap_dmapctl(&ctl_page);
341
342                 return rc;
343         }
344         if (token[0] == 'u') {
345                 if (type == LEVEL2) {
346                         *lblock = 0;
347                         return DISPLAY_DBMAP;
348                 }
349                 if (type == LEVEL1) {
350                         if (dmap_level > 1) {
351                                 *lblock = 1;
352                                 return DISPLAY_CPAGE;
353                         } else {
354                                 *lblock = 0;
355                                 return DISPLAY_DBMAP;
356                         }
357                 }
358                 if (type == LEVEL0) {
359                         if (dmap_level > 0) {
360                                 *lblock = L1PAGE(l1);
```

```
361                                     return DISPLAY_CPAGE;
362                             } else {
363                                     *lblock = 0;
364                                     return DISPLAY_DBMAP;
365                             }
366                     }
367             }
368             if ((token[0] == 'r') || (token[0] == 'l')) {
369                     if (type == LEVEL2) {
370                             fputs("Level 2 node has no siblings!\n", stderr);
371                             goto cpage_retry;
372                     }
373                     if (type == LEVEL1) {
374                             if (dmap_level < 2) {
375                                     fputs("Level 1 node has no siblings!\n", stderr);
376                                     goto cpage_retry;
377                             }
378                             if (token[0] == 'r') {
379                                     if (l1 == LPERCTL - 1) {
380                                             fputs("No right sibling!\n", stderr);
381                                             goto cpage_retry;
382                                     }
383                                     l1++;
384                             } else {
385                                     if (l1 == 0) {
386                                             fputs("No left sibling!\n", stderr);
387                                             goto cpage_retry;
388                                     }
389                                     l1--;
390                             }
391                             *lblock = L1PAGE(l1);
392                             return DISPLAY_CPAGE;
393                     }
394                     if (type == LEVEL0) {
395                             if (dmap_level < 1) {
396                                     fputs("Level 0 node has no siblings!\n", stderr);
397                                     goto cpage_retry;
398                             }
399                             if (token[0] == 'r') {
400                                     if (l0 == LPERCTL - 1) {
401                                             if ((dmap_level == 1) || (l1 == LPERCTL - 1)) {
402                                                     fputs("No right sibling!\n", stderr);
403                                                     goto cpage_retry;
404                                             }
405                                             l1++;
406                                             l0 = 0;
407                                     } else
408                                             l0++;
409                             } else {
410                                     if (l0 == 0) {
411                                             if (l1 == 0) {
412                                                     fputs("No left sibling!\n", stderr);
413                                                     goto cpage_retry;
414                                             }
415                                             l1--;
416                                             l0 = LPERCTL - 1;
417                                     } else
418                                             l0--;
419                             }
420                             *lblock = L0PAGE(l1, l0);
421                             return DISPLAY_CPAGE;
422                     }
423                     fprintf(stderr, "display_cpage: decode_pagenum returned type %d\n", type);
424                     return DMAP_EXIT;
425             }
426             if (token[0] == 'm') {
427                     field = m_parse(cmdline, 5, &token);
428                     if (field == 0)
429                             goto cpage_retry;
430
431                     switch (field) {
432                     case 1:
433                             ctl_page.nleafs = strtoul(token, 0, 0);
434                             break;
435                     case 2:
436                             ctl_page.l2nleafs = strtoul(token, 0, 0);
437                             break;
438                     case 3:
439                             ctl_page.leafidx = strtoul(token, 0, 0);
440                             break;
441                     case 4:
442                             ctl_page.height = strtoul(token, 0, 0);
443                             break;
444                     case 5:
445                             ctl_page.budmin = strtoul(token, 0, 0);
446                             break;
447                     }
448
449                     /* swap if on big endian machine */
450                     ujfs_swap_dmapctl(&ctl_page);
```

```
451
452                 if (xWrite(address, sizeof (struct dmapctl), (char *) &ctl_page)) {
453                         fputs("display_cpage: Error writing control page!\n", stderr);
454                         /* swap back if on big endian machine */
455                         ujfs_swap_dmapctl(&ctl_page);
456                         return DMAP_EXIT;
457                 }
458
459                 /* swap back if on big endian machine */
460                 ujfs_swap_dmapctl(&ctl_page);
461
462                 goto cpage_redisplay;
463         }
464         return DMAP_EXIT;
465 }
466
467 /*****************************************************************************
468  ************************  Example output of display_leaf()
469
470 Dmap Page at block 144
471
472 [1] nblocks:            8192            [8] tree.budmin:        5
473 [2] nfree:              7624            [9] tree.stree:         Hit <enter>
474 [3] start:              0               [10] tree.pad:          Not Displayed
475 [4] tree.nleafs:        256             [11] pad:               Not Displayed
476 [5] tree.l2nleafs:      8               [12] wmap:              Type 'w'
477 [6] tree.leafidx:       85              [13] pmap:              Type 'p'
478 [7] tree.height:        4
479
480  *****************************************************************************/
481
482 int display_leaf(struct dinode * bmap_inode, int64_t * lblock)
483 {
484         int64_t address;
485         int changed = 0;
486         char cmdline[80];
487         struct dmap d_map;
488         int field;
489         int rc;
490         char *token;
491         int dmap, l0, l1, type;
492
493         type = decode_pagenum(*lblock, &l1, &l0, &dmap);
494
495         if (ujfs_rwdaddr(fd, &address, bmap_inode, (*lblock) << dmap_l2bpp,
496                          GET, bsize) || xRead(address, sizeof (struct dmap), (char *) &d_map)) {
497                 fputs("display_leaf: Error reading dmap page!\n", stderr);
498                 return DMAP_EXIT;
499         }
500
501         /* swap if on big endian machine */
502         ujfs_swap_dmap(&d_map);
503
504       leaf_redisplay:
505         printf("\nDmap Page at block %lld\n\n", (long long) (address >> l2bsize));
506
507         printf("[1] nblocks:\t\t%d\t\t", d_map.nblocks);
508         printf("[8] tree.budmin:\t%d\n", d_map.tree.budmin);
509         printf("[2] nfree:\t\t%d\t\t", d_map.nfree);
510         printf("[9] tree.stree:\t\tHit <enter>\n");
511         printf("[3] start:\t\t%lld\t\t", (long long) d_map.start);
512         printf("[10] tree.pad:\t\tNot Displayed\n");
513         printf("[4] tree.nleafs:\t%d\t\t", d_map.tree.nleafs);
514         printf("[11] pad:\t\tNot Displayed\n");
515         printf("[5] tree.l2nleafs:\t%d\t\t", d_map.tree.l2nleafs);
516         printf("[12] wmap:\t\tType 'w'\n");
517         printf("[6] tree.leafidx:\t%d\t\t", d_map.tree.leafidx);
518         printf("[13] pmap:\t\tType 'p'\n");
519         printf("[7] tree.height:\t%d\n", d_map.tree.height);
520
521       leaf_retry:
522         fputs("[m]odify, [u]p, [r]ight or [l]eft, [w]map, [p]map, e[x]it, [s]tree > ", stdout);
523         fgets(cmdline, 80, stdin);
524         token = strtok(cmdline, "       \n");
525
526         if ((token == 0) || (token[0] == 's')) {
527                 rc = display_tree((struct dmapctl *) & d_map.tree, lblock, &changed);
528                 /* swap if on big endian machine */
529                 ujfs_swap_dmap(&d_map);
530
531                 if (changed && xWrite(address, sizeof (struct dmap), (char *) &d_map)) {
532                         fputs("display_leaf: Error writing dmap page!\n", stderr);
533                         /* swap back if on big endian machine */
534                         ujfs_swap_dmap(&d_map);
535                         return DMAP_EXIT;
536                 }
537
538                 /* swap back if on big endian machine */
539                 ujfs_swap_dmap(&d_map);
540
```

```
541                     return rc;
542             }
543             if (token[0] == 'u') {
544                     *lblock = L0PAGE(l1, l0);
545                     return DISPLAY_CPAGE;
546             }
547             if (token[0] == 'r') {
548                     if (dmap < LPERCTL - 1) {
549                             *lblock = DMAPPAGE(l1, l0, dmap + 1);
550                             return (DISPLAY_LEAF);
551                     }
552                     if (dmap_level > 0) {
553                             if (l0 < LPERCTL - 1) {
554                                     *lblock = DMAPPAGE(l1, l0 + 1, 0);
555                                     return (DISPLAY_LEAF);
556                             }
557                             if ((dmap_level == 2) && (l1 < LPERCTL - 1)) {
558                                     *lblock = DMAPPAGE(l1 + 1, 0, 0);
559                                     return (DISPLAY_LEAF);
560                             }
561                     }
562                     fputs("display_leaf: No right sibling.\n", stderr);
563                     goto leaf_retry;
564             }
565             if (token[0] == 'l') {
566                     if (dmap > 0) {
567                             *lblock = DMAPPAGE(l1, l0, dmap - 1);
568                             return (DISPLAY_LEAF);
569                     }
570                     if (dmap_level > 0) {
571                             if (l0 > 0) {
572                                     *lblock = DMAPPAGE(l1, l0 - 1, LPERCTL - 1);
573                                     return (DISPLAY_LEAF);
574                             }
575                             if ((dmap_level == 2) && (l1 > 0)) {
576                                     *lblock = DMAPPAGE(l1 - 1, LPERCTL - 1, LPERCTL - 1);
577                                     return (DISPLAY_LEAF);
578                             }
579                     }
580                     fputs("display_leaf: No left sibling.\n", stderr);
581                     goto leaf_retry;
582             }
583             if ((token[0] == 'p') || (token[0] == 'w')) {
584                     if (token[0] == 'p')
585                             rc = display_map(d_map.pmap, LPERDMAP);
586                     else
587                             rc = display_map(d_map.wmap, LPERDMAP);
588
589                     /* swap if on big endian machine */
590                     ujfs_swap_dmap(&d_map);
591
592                     if ((rc & XPEEK_CHANGED) && xWrite(address, sizeof (struct dmap), (char *) &d_map)) {
593                             fputs("display_leaf: Error writing dmap page!\n", stderr);
594                             /* swap back if on big endian machine */
595                             ujfs_swap_dmap(&d_map);
596                             return DMAP_EXIT;
597                     }
598
599                     /* swap back if on big endian machine */
600                     ujfs_swap_dmap(&d_map);
601
602                     if (rc & XPEEK_REDISPLAY)
603                             goto leaf_redisplay;
604                     return DMAP_EXIT;
605             }
606             if (token[0] != 'm')    /* assume 'x'     */
607                     return DMAP_EXIT;
608
609             field = m_parse(cmdline, 8, &token);
610             if (field == 0)
611                     goto leaf_retry;
612
613             switch (field) {
614             case 1:
615                     d_map.nblocks = strtoul(token, 0, 0);
616                     break;
617             case 2:
618                     d_map.nfree = strtoul(token, 0, 0);
619                     break;
620             case 3:
621                     d_map.start = strtoull(token, 0, 0);
622                     break;
623             case 4:
624                     d_map.tree.nleafs = strtoul(token, 0, 0);
625                     break;
626             case 5:
627                     d_map.tree.l2nleafs = strtoul(token, 0, 0);
628                     break;
629             case 6:
630                     d_map.tree.leafidx = strtoul(token, 0, 0);
```

```
                break;
        case 7:
                d_map.tree.height = strtoul(token, 0, 0);
                break;
        case 8:
                d_map.tree.budmin = strtoul(token, 0, 0);
                break;
        }
        goto leaf_redisplay;
}

/*****************************************************************************
 ***********************  Example output of display_leaf()

Level 4                              [ 0] 10
                              /-------/  \-------\
                     /-------/                   \-------\
            /-------/                                    \-------\
           /                                                     \
   [  0]  9            [  1]  8            [  2] 10            [  3] 10
         /\                  /\                  /\                  /\
        /    \              /    \              /    \              /    \
      /        \          /        \          /        \          /        \
    /            \      /            \      /            \      /            \
   6    8    8    9    8    8    8    8   10   -1    8    9   10   -1    9    8

    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15

 *****************************************************************************/

int display_tree(struct dmapctl *ctlpage, int64_t *lblock, int *changed)
{
        char cmdline[80];
        int i, j;
        int level;
        int index;
        char *token;

        /* tree_offset[height of tree - level] is offset of first element
           of that level */

        static int tree_offset[6] = { 0, 1, 5, 21, 85, 341 };
        int dmap, l0, l1, type;
        int offset;
        int top_index = 0;
        int top_level = ctlpage->height;

        if ((ctlpage->height < 1) || (ctlpage->height > 5)) {
                fprintf(stderr,
                        "display_tree is confused.  ctlpage->height = %d\n", ctlpage->height);
                return DMAP_EXIT;
        }

        type = decode_pagenum(*lblock, &l1, &l0, &dmap);

      redisplay_tree:
        offset = tree_offset[ctlpage->height - top_level];
        printf("\nLevel %d\t\t\t\t [%2d] %2d\n", top_level, top_index,
               ctlpage->stree[offset + top_index]);

        i = top_index << 2;
        level = top_level - 1;
        offset = tree_offset[ctlpage->height - level];
        printf("\t\t\t    /--------/ \\\--------\\\n");
        printf("\t\t\t   /--------/\t\t\t \\\--------\\\n");
        printf("\t\t  /--------/\t\t\t\t \\\--------\\\n");
        printf("\t\t /\t\t\t\t\t\t \\\n");
        printf("  [%3d] %2d\t     [%3d] %2d\t\t  [%3d] %2d\t     [%3d] %2d\n",
               i, ctlpage->stree[offset + i],
               i + 1, ctlpage->stree[offset + i + 1],
               i + 2, ctlpage->stree[offset + i + 2], i + 3, ctlpage->stree[offset + i + 3]);

        if (level > 0) {
                i = top_index << 4;
                level--;
                offset = tree_offset[ctlpage->height - level];
                printf("\t /\\\t\t    /\\ \t\t /\\\t\t    /\\\n");
                printf("    /   \\\t\t   /\t\\\t     /   \\\t\t  /\t\\\n");
                printf("   /\t    \\ \t /\t  \\\t    /\t     \\ \t /\t  \\\n");
                printf("  /\t\t\\    /\t   \\\t   /\t\t\\    /\t   \\\n");
                printf(" %2d", ctlpage->stree[offset + i]);
                for (j = i + 1; j < i + 16; j++)
                        printf("  %2d", ctlpage->stree[offset + j]);
                printf("\n\n%4d", i);
                for (j = i + 1; j < i + 16; j++)
                        printf(" %4d", j);
                printf("\n");
        }
      tree_retry:
        fputs("\n[b]ack, [d]escend, [g]oto, [m]odify, [r]ight, [l]eft, [u]p, e[x]it > ", stdout);
```

```
721             fgets(cmdline, 80, stdin);
722             token = strtok(cmdline, "       \n");
723             if ((token == 0) || (token[0] == 'x'))
724                     return DMAP_EXIT;
725             if (token[0] == 'l') {
726                     if (top_index == 0) {   /* Need to move to another page */
727                             fputs("For now, go [b]ack, then go left\n", stdout);
728                             goto tree_retry;
729                     }
730                     top_index--;
731                     goto redisplay_tree;
732             }
733             if (token[0] == 'r') {
734                     if (top_index == (1 << ((ctlpage->height - top_level) << 1)) - 1) {
735                             fputs("For now, [b]ack then go right\n", stdout);
736                             goto tree_retry;
737                     }
738                     top_index++;
739                     goto redisplay_tree;
740             }
741             if (token[0] == 'g') {
742                     token = strtok(0, "     \n");
743                     if (token == 0) {
744                             fputs("Please enter: level index > ", stdout);
745                             fgets(cmdline, 80, stdin);
746                             token = strtok(cmdline, "       \n");
747                             if (token == 0)
748                                     goto tree_retry;
749                     }
750                     level = strtoul(token, 0, 0);
751                     token = strtok(0, "     \n");
752                     if (token == 0) {
753                             fputs("Not enough arguments\n", stderr);
754                             goto tree_retry;
755                     }
756                     index = strtoul(token, 0, 0);
757                     if ((level < 0) || (level > ctlpage->height) || (index < 0) ||
758                         (index >= (1 << ((ctlpage->height - level) << 1)))) {
759                             fputs("Invalid level and/or index\n", stderr);
760                             goto tree_retry;
761                     }
762                     if ((level == 1) && ctlpage->height > 1) {
763                             level++;
764                             index >>= 2;
765                     } else if (level == 0) {
766                             if (ctlpage->height == 1) {     /* is this possible? */
767                                     level++;
768                                     index >>= 2;
769                             } else {
770                                     level += 2;
771                                     index >>= 4;
772                             }
773                     }
774                     top_level = level;
775                     top_index = index;
776                     goto redisplay_tree;
777             }
778             if (token[0] == 'u') {
779                     if (top_level == ctlpage->height) {     /* At top of tree */
780                             fputs("Already at top of tree.\n", stdout);
781                             goto tree_retry;
782                     }
783                     /* Move up one level */
784                     top_level++;
785                     top_index >>= 2;
786                     goto redisplay_tree;
787             }
788             if (token[0] == 'd') {
789                     if (type == DMAP) {
790                             fputs("[d]escend only valid for control pages\n", stderr);
791                             goto tree_retry;
792                     }
793                     token = strtok(0, "     \n");
794                     if (token == 0) {
795                             fputs("Please enter: leaf# > ", stdout);
796                             fgets(cmdline, 80, stdin);
797                             token = strtok(cmdline, "       \n");
798                             if (token == 0)
799                                     goto tree_retry;
800                     }
801                     index = strtoul(token, 0, 0);
802                     if ((index < 0) || (index >= ctlpage->nleafs)) {
803                             fputs("Invalid leaf index\n", stderr);
804                             goto tree_retry;
805                     }
806                     if (type == LEVEL0) {
807                             *lblock = DMAPPAGE(l1, l0, index);
808                             return DISPLAY_LEAF;
809                     }
810                     if (type == LEVEL1)
```

```
811                             *lblock = L0PAGE(l1, index);
812                     else            /* LEVEL2 */
813                             *lblock = L1PAGE(index);
814
815                     return DISPLAY_CPAGE;
816             }
817             if (token[0] != 'm') {  /* Assuming 'b' */
818                     if (type == DMAP)
819                             return DISPLAY_LEAF;
820                     else
821                             return DISPLAY_CPAGE;
822             }
823             token = strtok(0, "     \n");
824             if (token == 0) {
825                     fputs("Please enter: level index value > ", stdout);
826                     fgets(cmdline, 80, stdin);
827                     token = strtok(0, "     \n");
828                     if (token == 0)
829                             goto tree_retry;
830             }
831             level = strtol(token, 0, 0);
832             token = strtok(0, "     \n");
833             if (token == 0) {
834                     fputs("Not enough arguments!\n", stderr);
835                     goto tree_retry;
836             }
837             index = strtol(token, 0, 0);
838             token = strtok(0, "     \n");
839             if (token == 0) {
840                     fputs("Not enough arguments!\n", stderr);
841                     goto tree_retry;
842             }
843             if ((level < 0) || (level > ctlpage->height) || (index < 0) ||
844                 (index >= (1 << ((ctlpage->height - level) << 1)))) {
845                     fputs("Invalid level and/or index\n", stderr);
846                     goto tree_retry;
847             }
848             ctlpage->stree[tree_offset[ctlpage->height - level] + index] = strtol(token, 0, 0);
849             *changed = 1;
850             goto redisplay_tree;
851     }
852
853     int decode_pagenum(int64_t page, int *l1, int *l0, int *dmap)
854     {
855             int remainder;
856
857             if (page == 0)
858                     return -1;
859
860             if (page == 1)
861                     return LEVEL2;
862
863             *l1 = (page - 2) / L1FACTOR;
864             remainder = (page - 2) % L1FACTOR;
865             if (remainder == 0)
866                     return LEVEL1;
867
868             *l0 = (remainder - 1) / L0FACTOR;
869             remainder = (remainder - 1) % L0FACTOR;
870             if (remainder == 0)
871                     return LEVEL0;
872
873             *dmap = remainder - 1;
874             return DMAP;
875     }
876
877     int display_agfree(int64_t * agfree)
878     {
879             char cmdline[80];
880             int end;
881             int i;
882             int index;
883             int rc = XPEEK_OK;
884             int start = 0;
885             char *token;
886
887           agfree_display:
888             end = MIN(start + 64, MAXAG);
889             for (i = start; i < end; i += 4)
890                     printf("%3d 0x%016llx 0x%016llx 0x%016llx 0x%016llx\n", i, (long long) agfree[i],
891                             (long long) agfree[i + 1], (long long) agfree[i + 2],
892                             (long long) agfree[i + 3]);
893           agfree_retry:
894             fputs("display_agfree: [m]odify, [b]ack, e[x]it > ", stdout);
895             fgets(cmdline, 80, stdin);
896             token = strtok(cmdline, "       \n");
897             if (token == 0) {
898                     start = (end < MAXAG) ? end : 0;
899                     goto agfree_display;
900             }
```

```
901             if (token[0] == 'x')
902                     return rc;
903             if (token[0] != 'm') {  /* assuming 'b' */
904                     return (rc | XPEEK_REDISPLAY);
905             }
906
907             index = m_parse(cmdline, MAXAG − 1, &token);
908             if (index == 0)
909                     goto agfree_retry;
910
911             agfree[index] = strtoull(token, 0, 16);
912             rc = XPEEK_CHANGED;
913             goto agfree_display;
914     }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *   FUNCTION: Explain how to use the xpeek utility
 */
#include "xpeek.h"

void help()
{
        char *cmd;
        int cmd_len;

        cmd = strtok(0, " \t\n");          /* space & tab */
        if (strtok(0, " \t\n")) {
                fputs("help: called with too many arguments\n\n", stderr);
                return;
        }
        if (cmd == 0) {
                fputs("\t\tXpeek Commands\n\n", stdout);
                fputs("a[lter] <block> <offset> <hex string>\n", stdout);
                fputs("b[tree] <block> [<offset>]\n", stdout);
                fputs("cb[blfsck]\n", stdout);
                fputs("dir[ectory] <inode number> [<fileset>]\n", stdout);
                fputs("d[isplay] [<block> [<offset> [<format> [<count>]]]]\n", stdout);
                fputs("dm[ap]\n", stdout);
                fputs("dt[ree] <inode number> [<fileset>]\n", stdout);
                fputs("fsckw[sphdr]\n", stdout);
                fputs("h[elp] [<command>]\n", stdout);
                fputs("ia[g] [<IAG number>] [a | s | <fileset>]\n", stdout);
                fputs("i[node] [<inode number>] [a | s | <fileset>]\n", stdout);
                fputs("logs[uper]\n", stdout);
                fputs("q[uit]\n", stdout);
                fputs("se[t] [<variable> <value>]\n", stdout);
                fputs("su[perblock] [p | s]\n", stdout);
                fputs("s2p[erblock] [p | s]\n", stdout);
                fputs("u[nset] <variable>\n", stdout);
                fputs("xt[ree] <inode number> [<fileset>]\n\n", stdout);
                return;
        }
        cmd_len = strlen(cmd);
        if (strncmp(cmd, "alter", cmd_len) == 0) {
                fputs("a[lter] <block> <offset> <hex string>\n", stdout);
                fputs("\t<block>\t\tblock number (decimal)\n", stdout);
                fputs("\t<offset>\toffset within block (hex)\n", stdout);
                fputs("\t<hex string>\tstring of hex digits\n\n", stdout);
                fputs("alters disk data.  <hex string> should contain an even number of digits\n\n",
                      stdout);
        } else if (strncmp(cmd, "btree", cmd_len) == 0) {
                fputs("b[tree] <block> [<offset>]\n", stdout);
                fputs("\t<block>\t\tblock number (decimal)\n", stdout);
                fputs("\t<offset>\toffset within block (hex)\n\n", stdout);
                fputs("displays one node of a btree and enters a subcommand mode in which to\n",
                      stdout);
                fputs("navigate the btree.  Subcommands:\n", stdout);
                fputs("\tl\tvisit left sibling\n", stdout);
                fputs("\tm\tmodify current node\n", stdout);
                fputs("\tp\tvisit parent node\n", stdout);
                fputs("\tr\tvisit right sibling\n", stdout);
                fputs("\t[0−9]\tvisit the nth child node\n", stdout);
                fputs("\tx\texit subcommand mode\n\n", stdout);
        } else if (cmd_len > 1 && strncmp(cmd, "cbblfsck", cmd_len) == 0) {
                fputs("cb[blfsck]\n\n", stdout);
                fputs("Displays the area used by ClearBadBlockList \n", stdout);
                fputs("for communication with fsck.\n\n", stdout);
        } else if (cmd_len > 2 && strncmp(cmd, "directory", cmd_len) == 0) {
                fputs("dir[ectory] <inode number> [<fileset>]\n", stdout);
                fputs("\t<inode number>\tinode number of directory (decimal)\n", stdout);
                fputs("\t<fileset>\tfileset number, currently must be zero\n\n", stdout);
                fputs("Displays directory entries.  Subcommands\n", stdout);
                fputs("\tm\tmodify entries\n", stdout);
                fputs("\tx\texit subcommand mode\n\n", stdout);
        } else if (cmd_len > 1 && strncmp(cmd, "dtree", cmd_len) == 0) {
                fputs("dt[ree] <inode number> [<fileset>]\n", stdout);
                fputs("\t<inode number>\tinode number of directory (decimal)\n", stdout);
                fputs("\t<fileset>\tfileset number, currently must be zero\n\n", stdout);
```

```
91                  fputs
92                      ("Displays root of the directory btree and enters a subcommand mode in which to\n",
93                       stdout);
94                  fputs("navigate the btree.  Subcommands:\n", stdout);
95                  fputs("\tl\tvisit left sibling\n", stdout);
96                  fputs("\tm\tmodify current node\n", stdout);
97                  fputs("\tp\tvisit parent node (not parent directory)\n", stdout);
98                  fputs("\tr\tvisit right sibling\n", stdout);
99                  fputs("\t[0−9]\tvisit the nth child node\n", stdout);
100                 fputs("\tx\texit subcommand mode\n\n", stdout);
101         } else if (strncmp(cmd, "display", cmd_len) == 0) {
102                 fputs("d[isplay] [<block> [<offset> [<format> [<count>]]]]\n", stdout);
103                 fputs("\t<block>\t\tblock number (decimal)\n", stdout);
104                 fputs("\t<offset>\toffset within block (hex)\n", stdout);
105                 fputs("\t<format>\tformat in which to display data (see below)\n", stdout);
106                 fputs("\t<count>\t\tnumber of objects to display (decimal)\n", stdout);
107                 fputs
108                     ("displays data in a variety of formats.  Format may be one of the following:\n",
109                      stdout);
110                 fputs("\ta\tascii\n", stdout);
111                 fputs("\tb\tblock allocation map\n", stdout);
112                 fputs("\td\tdecimal\n", stdout);
113                 fputs("\ti\tinode\t\t\tstruct dinode\n", stdout);
114                 fputs("\tI\tinode allocation map\t\tixad_t\n", stdout);
115                 fputs("\ts\tsuperblock\t\t\tstruct superblock\n", stdout);
116                 fputs("\tx\thexadecimal\n", stdout);
117                 fputs("\tX\textent allocation descriptor\txad_t\n\n", stdout);
118         } else if (cmd_len > 1 && strncmp(cmd, "dmap", cmd_len) == 0) {
119                 fputs("dm[ap]\n", stdout);
120                 fputs("Displays Aggregate Disk (Block) Map\n\n", stdout);
121         } else if (cmd_len > 4 && strncmp(cmd, "fsckwsphdr", cmd_len) == 0) {
122                 fputs("fsckw[sphdr]\n\n", stdout);
123                 fputs("Displays the header of the fsck workspace in the aggregate.\n\n", stdout);
124         } else if (strncmp(cmd, "help", cmd_len) == 0) {
125                 fputs("h[elp] [<command>]\n", stdout);
126                 fputs("\t<command>\tcommand name\n\n", stdout);
127                 fputs("prints help text.  Lists all commands if no parameter\n\n", stdout);
128         } else if (cmd_len > 1 && strncmp(cmd, "iag", cmd_len) == 0) {
129                 fputs("ia[g] [<IAG number>] [a | <fileset>]\n", stdout);
130                 fputs("\t<IAG number>\tIAG number (decimal)\n", stdout);
131                 fputs("\ta\t\tuse primary aggragate inode table\n", stdout);
132                 fputs("\ts\t\tuse secondary aggragate inode table\n", stdout);
133                 fputs("\t<fileset>\tfileset number (currently must be zero)\n\n", stdout);
134                 fputs("Display iag information and enters subcommand mode.  Subcommands:\n",
135                       stdout);
136                 fputs("\te\tdisplay/modify inode extents map\n", stdout);
137                 fputs("\tm\tmodify iag\n", stdout);
138                 fputs("\tp\tdisplay/modify persistent map\n", stdout);
139                 fputs("\tw\tdisplay/modify working map\n\n", stdout);
140         } else if (strncmp(cmd, "inode", cmd_len) == 0) {
141                 fputs("i[node] [<inode number>] [a | <fileset>]\n", stdout);
142                 fputs("\t<inode number>\tinode number (decimal)\n", stdout);
143                 fputs("\ta\t\tuse primary aggragate inode table\n", stdout);
144                 fputs("\ts\t\tuse secondary aggragate inode table\n", stdout);
145                 fputs("\t<fileset>\tfileset number (currently must be zero)\n\n", stdout);
146                 fputs("Display inode information and enters subcommand mode.  Subcommands:\n",
147                       stdout);
148                 fputs("\ta\tdisplay/modify inode's ACL\n", stdout);
149                 fputs("\tm\tmodify inode\n", stdout);
150         } else if (cmd_len > 3 && strncmp(cmd, "logsuper", cmd_len) == 0) {
151                 fputs("logs[uper]\n\n", stdout);
152                 fputs("Displays the journal log superblock.\n\n", stdout);
153         } else if (strncmp(cmd, "quit", cmd_len) == 0) {
154                 fputs("q[uit]\n\n", stdout);
155                 fputs("exit xpeek\n\n", stdout);
156         } else if (cmd_len > 1 && strncmp(cmd, "set", cmd_len) == 0) {
157                 fputs("se[t] [<variable> <value>]\n", stdout);
158                 fputs("\t<variable>\tvariable to define.  First character must be alpha\n", stdout);
159                 fputs("\t<value>\t\tvalue to assign to variable.\n\n", stdout);
160                 fputs("Sets a user−defined variable.  With no arguments, set lists all\n", stdout);
161                 fputs("defined variables.\n", stdout);
162         } else if (cmd_len > 1 && strncmp(cmd, "superblock", cmd_len) == 0) {
163                 fputs("su[perblock] [p | s]\n", stdout);
164                 fputs("\tp\tDisplay primary superblock\n", stdout);
165                 fputs("\ts\tDisplay secondary superblock\n\n", stdout);
166                 fputs("Displays superblock data\n\n", stdout);
167         } else if (cmd_len > 2 && strncmp(cmd, "s2perblock", cmd_len) == 0) {
168                 fputs("su[perblock] [p | s]\n", stdout);
169                 fputs("\tp\tDisplay primary superblock\n", stdout);
170                 fputs("\ts\tDisplay secondary superblock\n\n", stdout);
171                 fputs("Displays alternate superblock data\n\n", stdout);
172         } else if (strncmp(cmd, "unset", cmd_len) == 0) {
173                 fputs("u[nset] <variable>\n\n", stdout);
174                 fputs("\t<variable>\tvariable to delete.\n\n", stdout);
175                 fputs("Deletes variables defined with set command\n\n", stdout);
176         } else if (cmd_len > 1 && strncmp(cmd, "xtree", cmd_len) == 0) {
177                 fputs("xt[ree] <inode number> [<fileset>]\n", stdout);
178                 fputs("\t<inode number>\tinode number of non−directory (decimal)\n", stdout);
179                 fputs("\t<fileset>\tfileset number, currently must be zero\n\n", stdout);
180                 fputs
```

```
181                         ("Displays root of the non−directory btree and enters a subcommand mode in which to\n",
182                          stdout);
183                 fputs("navigate the btree.  Subcommands:\n", stdout);
184                 fputs("\tl\tvisit left sibling\n", stdout);
185                 fputs("\tm\tmodify current node\n", stdout);
186                 fputs("\tp\tvisit parent node (not parent directory)\n", stdout);
187                 fputs("\tr\tvisit right sibling\n", stdout);
188                 fputs("\t[0−9]\tvisit the nth child node\n", stdout);
189                 fputs("\tx\texit subcommand mode\n\n", stdout);
190         } else
191                 fprintf(stderr, "help: Unknown command '%s'\n\n", cmd);
192 }
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *   io.c − I/O routines
 */
#include "xpeek.h"

/* libfs includes */
#include <devices.h>

int xRead(int64_t address, unsigned count, char *buffer)
{
        int64_t block_address;
        char *block_buffer;
        int64_t length;
        unsigned offset;

        offset = address & (bsize − 1);
        length = (offset + count + bsize − 1) & ~(bsize − 1);

        if ((offset == 0) & (length == count))
                return ujfs_rw_diskblocks(fd, address, count, buffer, GET);

        block_address = address − offset;
        block_buffer = (char *) malloc(length);
        if (block_buffer == 0)
                return 1;

        if (ujfs_rw_diskblocks(fd, block_address, length, block_buffer, GET)) {
                free(block_buffer);
                return 1;
        }
        memcpy(buffer, block_buffer + offset, count);
        free(block_buffer);
        return 0;
}

int xWrite(int64_t address, unsigned count, char *buffer)
{
        int64_t block_address;
        char *block_buffer;
        int64_t length;
        unsigned offset;

        offset = address & (bsize − 1);
        length = (offset + count + bsize − 1) & ~(bsize − 1);

        if ((offset == 0) & (length == count))
                return ujfs_rw_diskblocks(fd, address, count, buffer, PUT);

        block_address = address − offset;
        block_buffer = (char *) malloc(length);
        if (block_buffer == 0)
                return 1;

        if (ujfs_rw_diskblocks(fd, block_address, length, block_buffer, GET)) {
                free(block_buffer);
                return 1;
        }
        memcpy(block_buffer + offset, buffer, count);
        if (ujfs_rw_diskblocks(fd, block_address, length, block_buffer, PUT)) {
                free(block_buffer);
                return 1;
        }
        free(block_buffer);
        return 0;
}
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *       ui.c − User Interface routines
 */
#include "xpeek.h"

/*
        m_parse − parse parameters to 'm'odify subcommand

        NOTE:  Assumes last call to strtok() parsed "m" subcommand from
        command line.
*/
int m_parse(char *cmd_line, int n_fields, char **value)
{
        int field_number;
        char *token;

        token = strtok(0, "     \n");
        if (token == 0) {
                fputs("Please enter: field−number value > ", stdout);
                fgets(cmd_line, 80, stdin);
                token = strtok(cmd_line, "      \n");
                if (token == 0)
                        return 0;
        }
        field_number = strtol(token, 0, 0);
        if (field_number < 1 || field_number > n_fields) {
                fputs("Invalid field number\n", stderr);
                return 0;
        }
        *value = strtok(0, "    \n");
        if (*value == 0) {
                fputs("Not enough arguments\n", stderr);
                return 0;
        }
        if (strtok(0, "         \n")) {
                fputs("Too many arguments\n", stderr);
                return 0;
        }
        return field_number;
}
```

```
/*
 *   Copyright (c) International Business Machines Corp., 2000−2002
 *
 *   This program is free software;  you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY;  without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 *   the GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program;  if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 */
/*
 *   FUNCTION: common data & function prototypes
 */
#include <jfs_types.h>

/* system includes */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

/* JFS includes */
#include <jfs_dinode.h>
#include <jfs_imap.h>
#include <jfs_superblock.h>
#include <devices.h>

/* Defines */
#define AGGREGATE_2ND_I −1

#define XPEEK_OK        0x00
#define XPEEK_CHANGED   0x01
#define XPEEK_REDISPLAY 0x10
#define XPEEK_ERROR     −1

/* Global Data */
extern int bsize;
extern HFILE fd;
extern short l2bsize;

/* xpeek functions */

void alter(void);
void cbblfsck(void);
void directory(void);
void display(void);
void display_iag(struct iag *);
void display_inode(struct dinode *);
int display_super(struct superblock *);
void dmap(void);
void dtree(void);
void help(void);
int find_iag(unsigned iagnum, unsigned which_table, int64_t * address);
int find_inode(unsigned inum, unsigned which_table, int64_t * address);
void fsckwsphdr(void);
void iag(void);
void inode(void);
void logsuper(void);
int m_parse(char *, int, char **);
int more(void);
char prompt(char *);
void superblock(void);
void s2perblock(void);
void xtree(void);

int xRead(int64_t, unsigned, char *);
int xWrite(int64_t, unsigned, char *);

#define fputs(string,fd) { fputs(string,fd); fflush(fd); }
```

Table of Content