```
**********************************************************
   20071 Fri Jul 17 23:20:33 2015
new/usr/src/cmd/svr4pkg/hdrs/libinst.h
6063 pkgadd breaks with too many mountpoints
**********************************************************
______unchanged_portion_omitted_

 154 #if defined(__STDC__)
 155 #define __P(protos) protos
 156 #else   /* __STDC__ */
 157 #define __P(protos) ()
 158 #endif  /* __STDC__ */

 160 /* Common quit declaration used across many package commands */
 161 extern void     quit(int) __NORETURN;


 164 /* listmgr.c */
 165 extern int      bl_create __P((int count_per_block, int struct_size,
 166                     char *desc));
 167 extern char     *bl_next_avail __P((int list_handle));
 168 extern char     *bl_get_record __P((int list_handle, int recno));
 169 extern void     bl_free __P((int list_handle));
 170 extern int      ar_create __P((int count_per_block, int struct_size,
 171                     char *desc));
 172 extern char     **ar_next_avail __P((int list_handle));
 173 extern char     **ar_get_head __P((int list_handle));
 174 extern int      ar_delete __P((int list_handle, int index));
 175 extern void     ar_free __P((int list_handle));

 177 /* doulimit.c */
 178 extern int      set_ulimit __P((char *script, char *err_msg));
 179 extern int      clr_ulimit __P((void));
 180 extern int      assign_ulimit __P((char *fslimit));

 182 /* dryrun.c */
 183 extern void     set_continue_not_ok __P((void));
 184 extern int      continue_is_ok __P((void));
 185 extern int      in_dryrun_mode __P((void));
 186 extern int      in_continue_mode __P((void));
 187 extern void     init_dryrunfile __P((char *dr_dir));
 188 extern void     init_contfile __P((char *cn_dir));
 189 extern void     set_dr_exitmsg __P((char *value));
 190 extern void     set_dr_info __P((int type, int value));
 191 extern void     write_dryrun_file __P((struct cfextra **extlist));

 193 /* instvol.c */
 194 extern void     regfiles_free __P((void));

 196 /* lockinst.c */
 197 extern int      lockinst __P((char *util_name, char *pkg_name, char *place));
 198 extern void     lockupd __P((char *place));
 199 extern void     unlockinst __P((void));

 201 extern char     *pathdup __P((char *s));
 202 extern char     *pathalloc __P((int n));
 203 extern char     *fixpath __P((char *path));
 204 extern char     *get_info_basedir __P((void));
 205 extern char     *get_basedir __P((void));
 206 extern char     *get_client_basedir __P((void));
 207 extern int      set_basedirs __P((int reloc, char *adm_basedir,
 208                     char *pkginst, int nointeract));
 209 extern int      eval_path __P((char **server_ptr, char **client_ptr,
 210                     char **map_ptr, char *path));
 211 extern int      get_orig_offset __P((void));
 212 extern char     *get_inst_root __P((void));
```



```
 213 extern char     *get_mount_point __P((uint32_t n));
 214 extern char     *get_remote_path __P((uint32_t n));
 213 extern char     *get_mount_point __P((short n));
 214 extern char     *get_remote_path __P((short n));
 215 extern void     set_env_cbdir __P((void));
 216 extern int      set_inst_root __P((char *path));
 217 extern void     put_path_params __P((void));
 218 extern int      mkpath __P((char *p));
 219 extern void     mkbasedir __P((int flag, char *path));
 220 extern int      is_an_inst_root __P((void));
 221 extern int      is_a_basedir __P((void));
 222 extern int      is_a_cl_basedir __P((void));
 223 extern int      is_relocatable __P((void));
 224 extern char     *orig_path __P((char *path));
 225 extern char     *orig_path_ptr __P((char *path));
 226 extern char     *qreason __P((int caller, int retcode, int started,
 227                     int includeZonename));
 228 extern char     *qstrdup __P((char *s));
 229 extern char     *srcpath __P((char *d, char *p, int part, int nparts));
 230 extern char     *trans_srcp_pi __P((char *local_path));
 231 extern int      copyf __P((char *from, char *to, time_t mytime));
 232 extern int      copyFile __P((int, int, char *, char *, struct stat *, long));
 233 extern int      openLocal __P((char *a_path, int a_oflag, char *a_tmpdir));
 234 extern int      dockdeps __P((char *depfile, int removeFlag,
 235                     boolean_t a_preinstallCheck));
 236 extern int      finalck __P((struct cfent *ept, int attrchg, int contchg,
 237                     boolean_t a_warning));

 239 /* dockdeps.c */
 240 extern void setUpdate __P((void));
 241 extern int  isUpdate __P((void));

 243 /* mntinfo.c */
 244 extern int      get_mntinfo __P((int map_client, char *vfstab_file));
 245 extern uint32_t fsys __P((char *path));
 246 extern struct fstable *get_fs_entry __P((uint32_t n));
 245 extern short    fsys __P((char *path));
 246 extern struct fstable *get_fs_entry __P((short n));
 247 extern int      mount_client __P((void));
 248 extern int      unmount_client __P((void));
 249 extern uint32_t resolved_fsys __P((char *path));
 250 extern char     *get_server_host __P((uint32_t n));
 251 extern char     *server_map __P((char *path, uint32_t fsys_value));
 252 extern int      use_srvr_map __P((char *path, uint32_t *fsys_value));
 253 extern int      use_srvr_map_n __P((uint32_t n));
 254 extern int      is_fs_writeable __P((char *path, uint32_t *fsys_value));
 255 extern int      is_remote_fs __P((char *path, uint32_t *fsys_value));
 256 extern int      is_served __P((char *path, uint32_t *fsys_value));
 257 extern int      is_mounted __P((char *path, uint32_t *fsys_value));
 258 extern int      is_fs_writeable_n __P((uint32_t n));
 259 extern int      is_remote_fs_n __P((uint32_t n));
 260 extern int      is_served_n __P((uint32_t n));
 261 extern int      is_mounted_n __P((uint32_t n));
 262 extern fsblkcnt_t       get_blk_size_n __P((uint32_t n));
 263 extern fsblkcnt_t       get_frag_size_n __P((uint32_t n));
 264 extern fsblkcnt_t       get_blk_used_n __P((uint32_t n));
 265 extern fsblkcnt_t       get_blk_free_n __P((uint32_t n));
 266 extern fsblkcnt_t       get_inode_used_n __P((uint32_t n));
 267 extern fsblkcnt_t       get_inode_free_n __P((uint32_t n));
 268 extern void     set_blk_used_n __P((uint32_t n, fsblkcnt_t value));
 269 extern char     *get_source_name_n __P((uint32_t n));
 270 extern char     *get_fs_name_n __P((uint32_t n));
 249 extern short    resolved_fsys __P((char *path));
 250 extern char     *get_server_host __P((short n));
 251 extern char     *server_map __P((char *path, short fsys_value));
 252 extern int      use_srvr_map __P((char *path, short *fsys_value));
```

```
253 extern int      use_srvr_map_n __P((short n));
254 extern int      is_fs_writeable __P((char *path, short *fsys_value));
255 extern int      is_remote_fs __P((char *path, short *fsys_value));
256 extern int      is_served __P((char *path, short *fsys_value));
257 extern int      is_mounted __P((char *path, short *fsys_value));
258 extern int      is_fs_writeable_n __P((short n));
259 extern int      is_remote_fs_n __P((short n));
260 extern int      is_served_n __P((short n));
261 extern int      is_mounted_n __P((short n));
262 extern fsblkcnt_t       get_blk_size_n __P((short n));
263 extern fsblkcnt_t       get_frag_size_n __P((short n));
264 extern fsblkcnt_t       get_blk_used_n __P((short n));
265 extern fsblkcnt_t       get_blk_free_n __P((short n));
266 extern fsblkcnt_t       get_inode_used_n __P((short n));
267 extern fsblkcnt_t       get_inode_free_n __P((short n));
268 extern void     set_blk_used_n __P((short n, fsblkcnt_t value));
269 extern char     *get_source_name_n __P((short n));
270 extern char     *get_fs_name_n __P((short n));
271 extern int      load_fsentry __P((struct fstable *fs_entry, char *name,
272                     char *fstype, char *remote_name));
273 extern int      isreloc __P((char *pkginstdir));
274 extern int      is_local_host __P((char *hostname));
275 extern void     fs_tab_free __P((void));

277 /* pkgdbmerg.c */
278 extern int      pkgdbmerg __P((PKGserver server, VFP_T *tmpvfp,
279                     struct cfextra **extlist));
280 extern int      files_installed __P((void));

282 /* ocfile.c */
283 extern int      trunc_tcfile __P((int fd));
284 extern int      ocfile __P((PKGserver *serverp, VFP_T **tmpvfp,
285                         fsblkcnt_t map_blks));
286 extern int      swapcfile __P((PKGserver server, VFP_T **a_tmpvfp,
287                         char *pkginst, int dbchg));
288 extern int      set_cfdir __P((char *cfdir));
289 extern int      socfile __P((PKGserver *server, boolean_t quiet));
290 extern int      relslock __P((void));
291 extern int      pkgWlock __P((int verbose));
292 extern int      iscfile __P((void));
293 extern int      vcfile __P((void));

295 extern fsblkcnt_t       nblk __P((fsblkcnt_t size, ulong_t bsize,
296                         ulong_t frsize));
297 extern struct   cfent **procmap __P((VFP_T *vfp, int mapflag, char *ir));
298 extern void     repl_cfent __P((struct cfent *new, struct cfent *old));
299 extern struct   cfextra **pkgobjmap __P((VFP_T *vfp, int mapflag, char *ir));
300 extern void     pkgobjinit __P((void));
301 extern int      seed_pkgobjmap __P((struct cfextra *ext_entry, char *path,
302                     char *local));
303 extern int      init_pkgobjspace __P((void));

305 /* eptstat.c */
306 extern void     pinfo_free __P((void));
307 extern struct   pinfo *eptstat __P((struct cfent *entry, char *pkg, char c));

309 /* echo.c */
310 /*PRINTFLIKE1*/
311 extern void     echo __P((char *a_fmt, ...));
312 /*PRINTFLIKE1*/
313 extern void     echoDebug __P((char *a_fmt, ...));
314 extern boolean_t        echoGetFlag __P((void));
315 extern boolean_t        echoDebugGetFlag __P((void));
316 extern boolean_t        echoSetFlag __P((boolean_t a_debugFlag));
317 extern boolean_t        echoDebugSetFlag __P((boolean_t a_debugFlag));
```



```
319 /* ptext.c */
320 /*PRINTFLIKE2*/
321 extern void     ptext __P((FILE *fp, char *fmt, ...));

323 /* putparam.c */
324 extern void     putparam __P((char *param, char *value));
325 extern void     getuserlocale __P((void));
326 extern void     putuserlocale __P((void));
327 extern void     putConditionInfo __P((char *, char *));

329 /* setadmin.c */
330 extern void             setadminFile __P((char *file));
331 extern char             *setadminSetting __P((char *a_paramName,
332                                 char *a_paramValue));
333 extern char             *set_keystore_admin __P((void));
334 extern boolean_t        get_proxy_port_admin __P((char **, ushort_t *));
335 extern boolean_t        check_keystore_admin __P((char **));
336 extern int              web_ck_retries __P((void));
337 extern int              web_ck_timeout __P((void));
338 extern int              web_ck_authentication __P((void));

340 /* setlist.c */
341 extern char     *cl_iscript __P((int idx));
342 extern char     *cl_rscript __P((int idx));
343 extern void     find_CAS __P((int CAS_type, char *bin_ptr, char *inst_ptr));
344 extern int      setlist __P((struct cl_attr ***plist, char *slist));
345 extern void     addlist __P((struct cl_attr ***plist, char *item));
346 extern char     *cl_nam __P((int cl_idx));
347 extern char     *flex_device(char *device_name, int dev_ok);
348 extern int      cl_getn __P((void));
349 extern int      cl_idx __P((char *cl_nam));
350 extern void     cl_sets __P((char *slist));
351 extern void     cl_setl __P((struct cl_attr **cl_lst));
352 extern void     cl_putl __P((char *parm_name, struct cl_attr **list));
353 extern int      cl_deliscript __P((int i));
354 extern unsigned cl_svfy __P((int i));
355 extern unsigned cl_dvfy __P((int i));
356 extern unsigned cl_pthrel __P((int i));

358 /* passwd.c */
359 extern int      pkg_passphrase_cb __P((char *, int, int, void *));
360 extern void     set_passarg __P((char *));
361 extern void     set_prompt __P((char *));

363 /* fixpath.c */
364 extern void __P(export_client_env(char *));
365 extern void __P(set_partial_inst(void));
366 extern int __P(is_partial_inst(void));
367 extern void __P(set_depend_pkginfo_DB(boolean_t a_setting));
368 extern boolean_t __P(is_depend_pkginfo_DB(void));
369 extern void __P(disable_spool_create(void));
370 extern int __P(is_spool_create(void));

372 /* open_package_datastream.c */
373 extern boolean_t        open_package_datastream(int a_argc, char **a_argv,
374                                 char *a_spoolto, char *a_device,
375                                 int *r_repeat, char **r_idsName,
376                                 char *a_tmpdir, struct pkgdev *a_pkgdev,
377                                 int a_optind);

379 /* setup_temporary_directory.c */
380 extern boolean_t        setup_temporary_directory(char **r_dirname,
381                                 char *a_tmpdir, char *a_suffix);

383 /* unpack_package_from_stream.c */
384 extern boolean_t        unpack_package_from_stream(char *a_idsName,
```

```
385                                 char *a_pkginst, char *a_tempDir);

387 /* pkgops.c */

389 extern boolean_t        pkgAddPackageToGzonlyList(char *a_pkgInst,
390                                 char *a_rootPath);
391 extern void             pkgAddThisZonePackage(char *a_pkgInst);
392 extern boolean_t        pkgRemovePackageFromGzonlyList(char *a_rootPath,
393                                 char *a_pkgInst);
394 extern FILE             *pkgOpenInGzOnlyFile(char *a_rootPath);
395 extern void             pkginfoFree(struct pkginfo **r_info);
396 extern boolean_t        pkginfoIsPkgInstalled(struct pkginfo **r_pinfo,
397                                 char *a_pkgInst);
398 extern boolean_t        pkgIsPkgInGzOnly(char *a_rootPath, char *a_pkgInst);
399 extern boolean_t        pkgIsPkgInGzOnlyFP(FILE *a_fp, char *a_pkgInst);
400 extern boolean_t        pkginfoParamTruth(FILE *a_fp, char *a_param,
401                                 char *a_value, boolean_t a_default);
402 extern int              pkgGetPackageList(char ***r_pkgList, char **a_argv,
403                                 int a_optind, char *a_categories,
404                                 char **a_categoryList, struct pkgdev *a_pkgdev);
405 extern void             pkgLocateHighestInst(char *r_path, int r_pathLen,
406                                 char *r_pkgInst, int r_pkgInstLen,
407                                 char *a_rootPath, char *a_pkgInst);
408 extern boolean_t        pkgPackageIsThisZone(char *a_pkgInst);
409 extern char             *pkgGetGzOnlyPath(void);
410 extern boolean_t        pkgTestInstalled(char *a_packageName, char *a_rootPath);

412 /* depchk.c */

414 struct depckErrorRecord {
415         int     ier_numZones;
416         char    *ier_packageName;
417         char    **ier_zones;
418         char    **ier_values;
419 };
```

unchanged_portion_omitted_

```
**********************************************************
   35685 Fri Jul 17 23:20:33 2015
new/usr/src/cmd/svr4pkg/libinst/mntinfo.c
6063 pkgadd breaks with too many mountpoints
**********************************************************
```

_____*unchanged_portion_omitted_*

```
 217 /* This returns the hostname portion of a remote path. */
 218 char *
 219 get_server_host(uint32_t n)
 219 get_server_host(short n)
 220 {
 221         static char hostname[HOST_NM_LN], *host_end;

 223         if (fs_tab_used == 0) {
 224                 return ("unknown source");
 225         }

 227         if (n < fs_tab_used) {
 227         if (n >= 0 && n < fs_tab_used) {
 228                 (void) strcpy(hostname, fs_tab[n]->remote_name);
 229                 if ((host_end = strchr(hostname, ':')) == NULL) {
 230                         if ((strcmp(fs_tab[n]->fstype, MNTTYPE_AUTO)) == NULL)
 231                                 return ("automounter");
 232                         else
 233                                 return (fs_tab[n]->fstype);
 234                 } else {
 235                         *host_end = '\0';
 236                         return (hostname);
 237                 }
 238         }

 240         return ("unknown source");
 241 }
```

_____*unchanged_portion_omitted_*

```
 562 /*
 563  * This function maps path, on a loopback filesystem, back to the real server
 564  * filesystem. fsys_value is the fs_tab[] entry to which the loopback'd path is
 565  * mapped. This returns a pointer to a static area. If the result is needed
 566  * for further processing, it should be strdup()'d or something.
 567  */
 568 char *
 569 server_map(char *path, uint32_t fsys_value)
 569 server_map(char *path, short fsys_value)
 570 {
 571         static char server_construction[PATH_MAX];

 573         if (fs_tab_used == 0) {
 574                 (void) strcpy(server_construction, path);
 575         } else if (fsys_value < fs_tab_used) {
 575         } else if (fsys_value >= 0 && fsys_value < fs_tab_used) {
 576                 (void) snprintf(server_construction,
 577                         sizeof (server_construction),
 578                         "%s%s", fs_tab[fsys_value]->remote_name,
 579                         path+strlen(fs_tab[fsys_value]->name));
 580         } else {
 581                 (void) strcpy(server_construction, path);
 582         }

 584         return (server_construction);
 585 }
```

_____*unchanged_portion_omitted_*

```
1048 /*
1049  * Given a path, return the table index of the filesystem the file apparently
```



```
1050  * resides on. This doesn't put any time into resolving filesystems that
1051  * refer to other filesystems. It just returns the entry containing this
1052  * path.
1053  */
1054 uint32_t
1054 short
1055 fsys(char *path)
1056 {
1057         register int i;
1058         char    real_path[PATH_MAX];
1059         char    path_copy[PATH_MAX];
1060         char    *path2use;
1061         char    *cp;
1062         int     pathlen;
1063         boolean_t found = B_FALSE;

1065         /*
1066          * The loop below represents our best effort to identify real path of
1067          * a file, which doesn't need to exist. realpath() returns error for
1068          * nonexistent path, therefore we need to cut off trailing components
1069          * of path until we get path which exists and can be resolved by
1070          * realpath(). Lookup of "/dir/symlink/nonexistent-file" would fail
1071          * to resolve symlink without this.
1072          */
1073         (void) strlcpy(path_copy, path, PATH_MAX);
1074         for (cp = dirname(path_copy); strlen(cp) > 1; cp = dirname(cp)) {
1075                 if (realpath(cp, real_path) != NULL) {
1076                         found = B_TRUE;
1077                         break;
1078                 } else if (errno != ENOENT)
1079                         break;
1080         }
1081         if (found)
1082                 path2use = real_path;
1083         else
1084                 /* fall back to original path in case of unexpected failure */
1085                 path2use = path;

1087         pathlen = strlen(path2use);

1089         /*
1090          * The following algorithm scans the list of attached file systems
1091          * for the one containing path. At this point the file names in
1092          * fs_tab[] are sorted by decreasing length to facilitate the scan.
1093          * The first for() scans past all the file system names too short to
1094          * contain path. The second for() does the actual string comparison.
1095          * It tests first to assure that the comparison is against a complete
1096          * token by assuring that the end of the filesystem name aligns with
1097          * the end of a token in path2use (ie: '/' or NULL) then it does a
1098          * string compare. -- JST
1099          */

1101         if (fs_tab_used == 0) {
1102                 return (-1);
1103         }

1105         for (i = 0; i < fs_tab_used; i++)
1106                 if (fs_tab[i] == NULL)
1107                         continue;
1108                 else if (fs_tab[i]->namlen <= pathlen)
1109                         break;
1110         for (; i < fs_tab_used; i++) {
1111                 int fs_namelen;
1112                 char term_char;

1114                 if (fs_tab[i] == NULL)
```

```
1115                        continue;

1117                fs_namelen = fs_tab[i]->namlen;
1118                term_char = path2use[fs_namelen];

1120                /*
1121                 * If we're putting the file "/a/kernel" into the filesystem
1122                 * "/a", then fs_namelen == 2 and term_char == '/'. If, we're
1123                 * putting "/etc/termcap" into "/", fs_namelen == 1 and
1124                 * term_char (unfortunately) == 'e'. In the case of
1125                 * fs_namelen == 1, we check to make sure the filesystem is
1126                 * "/" and if it is, we have a guaranteed fit, otherwise we
1127                 * do the string compare. -- JST
1128                 */
1129                if ((fs_namelen == 1 && *(fs_tab[i]->name) == '/') ||
1130                    ((term_char == '/' || term_char == NULL) &&
1131                    strncmp(fs_tab[i]->name, path2use, fs_namelen) == 0))
1132                        return (i);
1133        }

1135        /*
1136         * It only gets here if the root filesystem is fundamentally corrupt.
1137         * (This can happen!)
1138         */
1139        progerr(ERR_FSYS_FELLOUT, path2use);

1141        return (-1);
1142 }

1144 /*
1145  * This function returns the entry in the fs_tab[] corresponding to the
1146  * actual filesystem of record. It won't return a loopback filesystem entry,
1147  * it will return the filesystem that the loopback filesystem is mounted
1148  * over.
1149  */
1150 uint32_t
1150 short
1151 resolved_fsys(char *path)
1152 {
1153        int i = -1;
1154        char path2use[PATH_MAX];

1156        (void) strcpy(path2use, path);

1158        /* If this isn't a "real" filesystem, resolve the map. */
1159        do {
1160                (void) strcpy(path2use, server_map(path2use, i));
1161                i = fsys(path2use);
1162        } while (fs_tab[i]->srvr_map);

1164        return (i);
1165 }

1167 /*
1168  * This function returns the srvr_map status based upon the fs_tab entry
1169  * number. This tells us if the server path constructed from the package
1170  * install root is really the target filesystem.
1171  */
1172 int
1173 use_srvr_map_n(uint32_t n)
1173 use_srvr_map_n(short n)
1174 {
1175        return ((int)fs_tab[n]->srvr_map);
1176 }

1178 /*
```



```
1179  * This function returns the mount status based upon the fs_tab entry
1180  * number. This tells us if there is any hope of gaining access
1181  * to this file system.
1182  */
1183 int
1184 is_mounted_n(uint32_t n)
1184 is_mounted_n(short n)
1185 {
1186        return ((int)fs_tab[n]->mounted);
1187 }

1189 /*
1190  * is_fs_writeable_n - given an fstab index, return 1
1191  *      if it's writeable, 0 if read-only.
1192  */
1193 int
1194 is_fs_writeable_n(uint32_t n)
1194 is_fs_writeable_n(short n)
1195 {
1196        /*
1197         * If the write access permissions haven't been confirmed, do that
1198         * now. Note that the only reason we need to do the special check is
1199         * in the case of an NFS mount (remote) because we can't determine if
1200         * root has access in any other way.
1201         */
1202        if (fs_tab[n]->remote && fs_tab[n]->mounted &&
1203            !fs_tab[n]->write_tested) {
1204                if (fs_tab[n]->writeable && !really_write(fs_tab[n]->name))
1205                        fs_tab[n]->writeable = 0;       /* not really */

1207                fs_tab[n]->write_tested = 1;    /* confirmed */
1208        }

1210        return ((int)fs_tab[n]->writeable);
1211 }

1213 /*
1214  * is_remote_fs_n - given an fstab index, return 1
1215  *      if it's a remote filesystem, 0 if local.
1216  *
1217  *      Note: Upon entry, a valid fsys() is required.
1218  */
1219 int
1220 is_remote_fs_n(uint32_t n)
1220 is_remote_fs_n(short n)
1221 {
1222        return ((int)fs_tab[n]->remote);
1223 }

1225 /* index-driven is_served() */
1226 int
1227 is_served_n(uint32_t n)
1227 is_served_n(short n)
1228 {
1229        return ((int)fs_tab[n]->served);
1230 }

1232 /*
1233  * This returns the number of blocks available on the indicated filesystem.
1234  *
1235  *      Note: Upon entry, a valid fsys() is required.
1236  */
1237 fsblkcnt_t
1238 get_blk_free_n(uint32_t n)
1238 get_blk_free_n(short n)
1239 {
```

```
1240         return (fs_tab[n]->bfree);
1241 }

1243 /*
1244  * This returns the number of blocks being used on the indicated filesystem.
1245  *
1246  *      Note: Upon entry, a valid fsys() is required.
1247  */
1248 fsblkcnt_t
1249 get_blk_used_n(uint32_t n)
1249 get_blk_used_n(short n)
1250 {
1251         return (fs_tab[n]->bused);
1252 }

1254 /*
1255  * This returns the number of inodes available on the indicated filesystem.
1256  *
1257  *      Note: Upon entry, a valid fsys() is required.
1258  */
1259 fsblkcnt_t
1260 get_inode_free_n(uint32_t n)
1260 get_inode_free_n(short n)
1261 {
1262         return (fs_tab[n]->ffree);
1263 }

1265 /*
1266  * This returns the number of inodes being used on the indicated filesystem.
1267  *
1268  *      Note: Upon entry, a valid fsys() is required.
1269  */
1270 fsblkcnt_t
1271 get_inode_used_n(uint32_t n)
1271 get_inode_used_n(short n)
1272 {
1273         return (fs_tab[n]->fused);
1274 }

1276 /*
1277  * Sets the number of blocks being used on the indicated filesystem.
1278  *
1279  *      Note: Upon entry, a valid fsys() is required.
1280  */
1281 void
1282 set_blk_used_n(uint32_t n, fsblkcnt_t value)
1282 set_blk_used_n(short n, fsblkcnt_t value)
1283 {
1284         fs_tab[n]->bused = value;
1285 }

1287 /* Get the filesystem block size. */
1288 fsblkcnt_t
1289 get_blk_size_n(uint32_t n)
1289 get_blk_size_n(short n)
1290 {
1291         return (fs_tab[n]->bsize);
1292 }

1294 /* Get the filesystem fragment size. */
1295 fsblkcnt_t
1296 get_frag_size_n(uint32_t n)
1296 get_frag_size_n(short n)
1297 {
1298         return (fs_tab[n]->bsize);
1299 }
```



```
1301 /*
1302  * This returns the name of the indicated filesystem.
1303  */
1304 char *
1305 get_fs_name_n(uint32_t n)
1305 get_fs_name_n(short n)
1306 {
1307         if (fs_tab_used == 0) {
1308                 return (NULL);
1309         } else if (n >= fs_tab_used) {
1310                 return (NULL);
1311         } else {
1312                 return (fs_tab[n]->name);
1313         }
1314 }

1316 /*
1317  * This returns the remote name of the indicated filesystem.
1318  *
1319  *      Note: Upon entry, a valid fsys() is required.
1320  */
1321 char *
1322 get_source_name_n(uint32_t n)
1322 get_source_name_n(short n)
1323 {
1324         return (fs_tab[n]->remote_name);
1325 }

1327 /*
1328  * This function returns the srvr_map status based upon the path.
1329  */
1330 int
1331 use_srvr_map(char *path, uint32_t *fsys_value)
1331 use_srvr_map(char *path, short *fsys_value)
1332 {
1333         if (*fsys_value == BADFSYS)
1334                 *fsys_value = fsys(path);

1336         return (use_srvr_map_n(*fsys_value));
1337 }

1339 /*
1340  * This function returns the mount status based upon the path.
1341  */
1342 int
1343 is_mounted(char *path, uint32_t *fsys_value)
1343 is_mounted(char *path, short *fsys_value)
1344 {
1345         if (*fsys_value == BADFSYS)
1346                 *fsys_value = fsys(path);

1348         return (is_mounted_n(*fsys_value));
1349 }

1351 /*
1352  * is_fs_writeable - given a cfent entry, return 1
1353  *      if it's writeable, 0 if read-only.
1354  *
1355  *      Note: Upon exit, a valid fsys() is guaranteed. This is
1356  *      an interface requirement.
1357  */
1358 int
1359 is_fs_writeable(char *path, uint32_t *fsys_value)
1359 is_fs_writeable(char *path, short *fsys_value)
1360 {
```

```
1361         if (*fsys_value == BADFSYS)
1362                 *fsys_value = fsys(path);
1363
1364         return (is_fs_writeable_n(*fsys_value));
1365 }
1366
1367 /*
1368  * is_remote_fs - given a cfent entry, return 1
1369  *      if it's a remote filesystem, 0 if local.
1370  *
1371  *      Also Note: Upon exit, a valid fsys() is guaranteed. This is
1372  *      an interface requirement.
1373  */
1374 int
1375 is_remote_fs(char *path, uint32_t *fsys_value)
1375 is_remote_fs(char *path, short *fsys_value)
1376 {
1377         if (*fsys_value == BADFSYS)
1378                 *fsys_value = fsys(path);
1379
1380         return (is_remote_fs_n(*fsys_value));
1381 }
1382
1383 /*
1384  * This function returns the served status of the filesystem. Served means a
1385  * client is getting this file from a server and it is not writeable by the
1386  * client. It has nothing to do with whether or not this particular operation
1387  * (eg: pkgadd or pkgrm) will be writing to it.
1388  */
1389 int
1390 is_served(char *path, uint32_t *fsys_value)
1390 is_served(char *path, short *fsys_value)
1391 {
1392         if (*fsys_value == BADFSYS)
1393                 *fsys_value = fsys(path);
1394
1395         return (is_served_n(*fsys_value));
1396 }
1397
1398 /*
1399  * get_remote_path - given a filesystem table index, return the
1400  *      path of the filesystem on the remote system.  Otherwise,
1401  *      return NULL if it's a local filesystem.
1402  */
1403 char *
1404 get_remote_path(uint32_t n)
1404 get_remote_path(short n)
1405 {
1406         char    *p;
1407
1408         if (!is_remote_fs_n(n))
1409                 return (NULL);  /* local */
1410         p = strchr(fs_tab[n]->remote_name, ':');
1411         if (!p)
1412                 p = fs_tab[n]->remote_name;     /* Loopback */
1413         else
1414                 p++;    /* remote */
1415         return (p);
1416 }
1417
1418 /*
1419  * get_mount_point - given a filesystem table index, return the
1420  *      path of the mount point.  Otherwise,
1421  *      return NULL if it's a local filesystem.
1422  */
1423 char *
```



```
1424 get_mount_point(uint32_t n)
1424 get_mount_point(short n)
1425 {
1426         if (!is_remote_fs_n(n))
1427                 return (NULL);  /* local */
1428         return (fs_tab[n]->name);
1429 }
1430
1431 struct fstable *
1432 get_fs_entry(uint32_t n)
1432 get_fs_entry(short n)
1433 {
1434         if (fs_tab_used == 0) {
1435                 return (NULL);
1436         } else if (n >= fs_tab_used) {
1437                 return (NULL);
1438         } else {
1439                 return (fs_tab[n]);
1440         }
1441 }
```

_____***unchanged_portion_omitted_***

****
3133 Fri Jul 17 23:20:33 2015
new/usr/src/lib/libpkg/common/cfext.h
6063 pkgadd breaks with too many mountpoints
****
_____unchanged_portion_omitted_

```
 55 /*
 56  * This is information required by pkgadd for fast operation. A
 57  * cfextra struct is tagged to each cfent structure requiring
 58  * processing. This is how we avoid some unneeded repetition. The
 59  * entries incorporating the word 'local' refer to the path that
 60  * gets us to the delivered package file. In other words, to install
 61  * a file we usually copy from 'local' to 'path' below. In the case
 62  * of a link, where no actual copying takes place, local is the source
 63  * of the link. Note that environment variables are not evaluated in
 64  * the locals unless they are links since the literal path is how
 65  * pkgadd finds the entry under the reloc directory.
 66  */
 67 struct cfextra {
 68         struct cfent cf_ent;    /* basic contents file entry */
 69         struct mergstat mstat;  /* merge status for installs */
 70         uint32_t   fsys_value; /* fstab[] entry index */
 71         uint32_t   fsys_base;  /* actual base filesystem in fs_tab[] */
 70         short   fsys_value; /* fstab[] entry index */
 71         short   fsys_base;  /* actual base filesystem in fs_tab[] */
 72         char    *client_path;   /* the client-relative path */
 73         char    *server_path;   /* the server-relative path */
 74         char    *map_path;  /* as read from the pkgmap */
 75         char    *client_local;  /* client_relative local */
 76         char    *server_local;  /* server relative local */
 77 };
```

_____unchanged_portion_omitted_