```
AppKeyMap.o/AlterAppKeyMap                          AppKeyMap.o/AlterAppKeyMap

   NAME
       AlterAppKeyMap -- Changes keyboard mappings for a keymap

   SYNOPSIS
       AlterAppKeyMap(mykeymap, keyarray, arrayentries);

       void AlterAppKeyMap(struct KeyMap *,
                          struct MyKey *,
                          UWORD);

   FUNCTION
       Using an array of MyKey structures (which describe a series of
       rawkeys), this functionchanges the rawkey's corresponding
       mapping values in mykeymap.  Only call this function on
       a private copy of a KeyMap (including its data).

   INPUTS
       mykeymap = pointer to a KeyMap structure.  AlterAppKeyMap() will
         make changes to this KeyMap's data, so this KeyMap should
         be a private copy of a KeyMap (including the data the KeyMap
         points to).

       keyarray  = pointer to an array of struct MyKey (from
         appkeymap.h):

           struct MyKey
           {
             UBYTE RawCode;/* RawCode of character to change      */
                          /* in the KeyMap.  Correponds to the   */
                          /* Lo/HiKeyMap structures from the     */
                          /* KeyMap structure.       */
             UBYTE MapType;/* The Lo/HiKeyMapTypes field.        */
             UBYTE Capsable;   /* This TRUE/FALSE state of this bit   */
                               /* gets translated to the correspond-  */
                               /* bit in KeyMap.Lo/HiCapsable.        */
             UBYTE Repeatable; /* This TRUE/FALSE state of this bit   */
                               /* gets translated to the correspond-  */
                               /* bit in KeyMap.Lo/HiRepeatable.      */
             ULONG Map;   /* Map data for key.  This points to   */
                          /* data for the rawkey.  Its format    */
                          /* depends on the key type. This field */
                          /* correponds to KeyMap.Lo/HiKeyMap.   */
           };

        arrayentries = The number of MyKey entries in keyarray.


   RESULT
       For each MyKey entry in keyarray, AlterKeyMap() finds the correpsonding
       raw key      entry in mykeymap and changes the data to match the entry
       in keyarray.  AlterKeyMap() makes changes directly to the KeyMap's
       data, so don't call this function on a system copy of a keymap, make
       a copy of it.

       Note that the console.device's CD_ASKKEYMAP only copies a KeyMap
       structure, which is only a set of pointers.  If you want to customize
       a copy of a keymap, you also have to duplicate the data that
       the KeyMap references.  If you do not duplicate the keymap data when
       customizing a keymap, you'll write over data that many keymaps are
       currently using.

       This function assumes that the keymap passed to it was duplicated
       using the CreateAppKeyMap() function.  CreateAppKeyMap() puts
       the keymap tables in a specific order and AlterAppKeyMap()
       expects to find the tables in that order.  Only call AlterAppKeyMap()
       on keymaps duplicated with CreateAppKeyMap().

   BUGS

   SEE ALSO
       console.device/CD_ASKKEYMAP console.device/CD_SETKEYMAP "appkeymap.h"
       AppKeyMap.o/CreateAppKeyMap() AppKeyMap.o/DeleteAppKeyMap()
```

```
AppKeyMap.o/CreateAppKeyMap                          AppKeyMap.o/CreateAppKeyMap

   NAME
       CreateAppKeyMap -- Create a new KeyMap by duplicating an existing one.

   SYNOPSIS
       newkeymap = CreateAppKeyMap(origkeymap);

       struct KeyMap *CreateAppKeyMap(struct KeyMap *);

   FUNCTION
       This function accepts a pointer to a KeyMap structure and duplicates it.
       CreateAppKeyMap() allocates the memory for a KeyMap structure and all
       the tables associated with that keymap.

   INPUTS
       origkeymap = points to a KeyMap to duplicate.

   RESULT
       If successful, this function returns a pointer to a duplicate of
       origkeymap.  CreateAppKeyMap() duplicates all of the tables that
       origkeymap references, so an application can make changes to the
       duplicate.  If CreateAppKeyMap() fails, it returns NULL.  The
       DeleteAppKeyMap() function deallocates the resources allocated
       by CreateAppKeyMap().

       CreateAppKeyMap() places the duplicate tables in an order so
       that a "Lo" table (for example, KeyMap.km_LoKeyMap) is immediately
       followed by the corresponding "Hi" table (KeyMap.km_HiKeyMap).
       This allows application to reference the two tables as one using
       the raw key value as an index.

   BUGS

   SEE ALSO
       console.device/CD_ASKKEYMAP console.device/CD_SETKEYMAP "appkeymap.h"
       AppKeyMap.o/AlterAppKeyMap() AppKeyMap.o/DeleteAppKeyMap()
```

```
AppKeyMap.o/DeleteAppKeyMap                          AppKeyMap.o/DeleteAppKeyMap

   NAME
       DeleteAppKeyMap -- Relinquish the resources allocated by
                          CreateAppKeyMap.

   SYNOPSIS
       DeleteAppKeyMap(mykeymap);

       void DeleteAppKeyMap(struct KeyMap *);

   FUNCTION
       This function accepts a pointer to a keymap allocated by
       CreateAppKeyMap() and deallocates the resources allocated
       by that function.

   INPUTS
       mykeymap = points to a KeyMap to deallocate.

   RESULT
       Frees memory previously used by mykeymap and its associated tables.

   BUGS

   SEE ALSO
       console.device/CD_ASKKEYMAP console.device/CD_SETKEYMAP "appkeymap.h"
       AppKeyMap.o/AlterAppKeyMap() AppKeyMap.o/CreateAppKeyMap()
```

AppKeyMap.h

```
#define UP_KEY      0x4C   /* Raw keys which are the same on all keyboards */
#define DOWN_KEY    0x4D
#define RIGHT_KEY   0x4E
#define LEFT_KEY    0x4F

#define F1_KEY      0x50
#define F2_KEY      0x51
#define F3_KEY      0x52
#define F4_KEY      0x53
#define F5_KEY      0x54
#define F6_KEY      0x55
#define F7_KEY      0x56
#define F8_KEY      0x57
#define F9_KEY      0x58
#define F10_KEY     0x59

#define N0_KEY      0x0F
#define N1_KEY      0x1D
#define N2_KEY      0x1E
#define N3_KEY      0x1F
#define N4_KEY      0x2D
#define N5_KEY      0x2E
#define N6_KEY      0x2F
#define N7_KEY      0x3D
#define N8_KEY      0x3E
#define N9_KEY      0x3F

#define NPERIOD_KEY 0x3C
#define NOPAREN_KEY 0x5A
#define NCPAREN_KEY 0x5B
#define NSLASH_KEY  0x5C
#define NASTER_KEY  0x5D
#define NMINUS_KEY  0x4A
#define NPLUS_KEY   0x5E
#define NENTER_KEY  0x43

/* Sizes of the "Lo" and "Hi" KeyMap tables */

#define LO_MAP_SIZE  64   /* Notice that the Lo and Hi tables are NOT the same */
#define HI_MAP_SIZE  56   /* size!  The Lo tables have eight more entries than */
#define LO_TYPE_SIZE 64   /* the Hi tables.  The RKM is incorrect in saying    */
#define HI_TYPE_SIZE 56   /* that the Lo and Hi tables are the same size.      */
#define LO_CAPS_SIZE 8     /* 8 bytes x 8 bits-per-byte = */
#define HI_CAPS_SIZE 7     /* 7 bytes x 8 bits-per-byte = */
#define LO_REPS_SIZE 8     /* 64 bit-wide entries.        */
#define HI_REPS_SIZE 7     /* 56 bit-wide entries.        */

void AlterAppKeyMap (struct KeyMap *,
                  struct MyKey *,
                  UWORD);
struct KeyMap *CreateAppKeyMap (struct KeyMap *);
void DeleteAppKeyMap (struct KeyMap *);
UBYTE FindMaxDeadKey(struct KeyMap *);

struct MyKey /* AlterAppKeyMap() expects an array of MyKey structures. */
             /* This structure contains all the information needed to  */
             /* redefine a key mapping.                                 */
{
  UBYTE RawCode;
  UBYTE MapType;    /* Qualifier flags and key type. */
  UBYTE Capsable;   /* Is this key Capsable?         */
  UBYTE Repeatable; /* Is this key Repeatable?       */
  ULONG Map;        /* This field either points to some key data or */
};                  /* contains a four-byte table of ANSI values.   */

struct KeyMapArrays  /* We allocate each Lo and Hi table pair as a single array */
{
  ULONG LHKeyMap[LO_MAP_SIZE << 1];
  UBYTE LHKeyMapTypes[LO_TYPE_SIZE << 1];
  UBYTE LHCapsable[LO_CAPS_SIZE << 1];
  UBYTE LHRepeatable[LO_REPS_SIZE << 1];
};
```

AppKeyMap.c

```
;/* appkeymap.c link module - Execute me to compile me with SAS C 6.2
sc data=far nominc strmer streq nostkchk saveds ign=73 appkeymap.c
quit
*/

/* appkeymap.c - subroutines to create/delete an application keymap
 *               by modifying a copy of a keymap
 *
 * The modifications made to the keymap are controlled by the "mykeys" array.
 * To create more complex keymappings, see the Rom Kernel Manual Libraries
 * keymap library chapter (p 821 et al).
 *
 * NOTE: disabling all numeric pad keys creates a good keymap for use
 * with either keymap.library MapANSI() or commodities InvertString().
 * If you used a default keymap with the above functions, numeric keypad
 * raw key values would be returned for keys which are available with
 * fewer keypresses on numeric pad than on the normal keyboard.
 * It is generally preferable to have the normal keyboard raw values
 * since many applications attach special meanings to numeric pad keys.
 */

#include <exec/types.h>
#include <exec/memory.h>
#include <devices/console.h>
#include <devices/keymap.h>

#include <clib/exec_protos.h>
#include <clib/keymap_protos.h>
#include <clib/alib_protos.h>

#include "appkeymap.h"

/* local functions */
static void CopyKeyMap (struct KeyMap *source, struct KeyMap *dest);


struct KeyMap *
CreateAppKeyMap (struct KeyMap *defkeymap)
{
  struct KeyMap *appkeymap = NULL;
  struct KeyMapArrays *karrays;
  ULONG keymapsize;

                              /* Because of the way this code allocates memory,  */
                              /* it must make sure the KeyMap structure will take*/
                              /* up an even amount of bytes (so the tables that  */
                              /* follow it will be word aligned).  This adds one */
                              /* to the size of the KeyMap structure's size if   */
                              /* necessary.  This is only for future compatibil- */
                              /* ity as a KeyMap structure is currently even.    */
  keymapsize = (sizeof(struct KeyMap)+1)&0x0000FFFE;

                              /* Allocate the space for the clone of the KeyMap. */
                              /* This includes space for the KeyMap structure    */
                              /* which is immediately followed by space for the  */
                              /* keymapping tables.                              */
  if (appkeymap = (struct KeyMap *)
          AllocVec (sizeof (struct KeyMap) + sizeof (struct KeyMapArrays),
                    MEMF_CLEAR|MEMF_PUBLIC) )
  {
                /* Initialize the pointer to the KeyMapArrays structure. These   */
                /* arrays are organized so that the "Lo" tables are immediately  */
                /* followed by the "Hi" tables, so you don't have to treat the   */
                /* Lo and Hi maps separately.  This allows AlterKeyMap() to find */
                /* Hi map raw key entries by use the raw key value as an index   */
                /* into the Lo map tables.                                       */
    karrays = (struct KeyMapArrays *) ((ULONG)appkeymap + keymapsize);

                /* Initialize the appkeymap fields to point to the KeyMapArrays. */
                /* Each LH array contains the Lo table followed by the Hi table. */
    appkeymap->km_LoKeyMap      = &karrays->LHKeyMap[0];
```

```
    appkeymap->km_HiKeyMap      = &karrays->LHKeyMap[LO_MAP_SIZE];
    appkeymap->km_LoKeyMapTypes = &karrays->LHKeyMapTypes[0];
    appkeymap->km_HiKeyMapTypes = &karrays->LHKeyMapTypes[LO_TYPE_SIZE];
    appkeymap->km_LoCapsable    = &karrays->LHCapsable[0];
    appkeymap->km_HiCapsable    = &karrays->LHCapsable[LO_CAPS_SIZE];
    appkeymap->km_LoRepeatable  = &karrays->LHRepeatable[0];
    appkeymap->km_HiRepeatable  = &karrays->LHRepeatable[LO_REPS_SIZE];


    CopyKeyMap (defkeymap, appkeymap);     /* Copy the keymap arrays to appkeymap. */
  }

  return (appkeymap);
}


void
DeleteAppKeyMap (struct KeyMap *appkeymap)
{
  if (appkeymap)
    FreeVec (appkeymap);
}


void
AlterAppKeyMap (struct KeyMap *appkeymap, struct MyKey *mykeys, UWORD mykeycount)
{
  ULONG *keymappings;
  UBYTE *keymaptypes, *capsables, *repeatables;
  UBYTE rawkeynum;
  int i, bytenum, bitnum;


/* AlterAppKeyMap() assumes that the tables of the keymap passed to it are */
/* contiguous from Lo through Hi. This allows AlterAppKeyMap() to directly */
/* reference any key using its raw key code as an index.  This also limits */
/* an application to using AlterAppKeyMap() to modify only keymap created  */
/* by CreateAppKeyMap().                                                   */


  keymappings = appkeymap->km_LoKeyMap;
  keymaptypes = appkeymap->km_LoKeyMapTypes;
  capsables   = appkeymap->km_LoCapsable;
  repeatables = appkeymap->km_LoRepeatable;

  for (i = 0; i < mykeycount; i++)
  {

                                    /* Because we allocated each of our Lo and Hi    */
    rawkeynum = mykeys[i].RawCode;  /* array pairs as sequential memory, we can use  */
                                    /* the RAWKEY values directly  to index into our */
                                    /* sequential Lo/Hi array.                       */

    keymaptypes[rawkeynum] = mykeys[i].MapType;
    keymappings[rawkeynum] = mykeys[i].Map;

    bytenum = rawkeynum >> 3; /* These are for the Capsable + Repeatable bit tables. */
    bitnum = rawkeynum % 8;   /*    Careful--There's only a 1 bit entry per raw key! */

    if (mykeys[i].Capsable)
      capsables[bytenum] |= (1 << bitnum);          /* If capsable, set bit, else... */
    else
      capsables[bytenum] &= (~(1 << bitnum));       /*             ...clear the bit. */

    if (mykeys[i].Repeatable)
      repeatables[bytenum] |= (1 << bitnum);      /* If repeatable, set bit, else... */
    else
      repeatables[bytenum] &= (~(1 << bitnum));   /*               ...clear the bit. */
  }
}
```

```
static void
CopyKeyMap (struct KeyMap *source, struct KeyMap *dest)
{
  UBYTE *bb;
  ULONG *ll;
  int i;

  for (i = 0, ll = source->km_LoKeyMap; i < LO_MAP_SIZE; i++)
    dest->km_LoKeyMap[i] = *ll++;
  for (i = 0, ll = source->km_HiKeyMap; i < HI_MAP_SIZE; i++)
    dest->km_HiKeyMap[i] = *ll++;

  for (i = 0, bb = source->km_LoKeyMapTypes; i < LO_TYPE_SIZE; i++)
    dest->km_LoKeyMapTypes[i] = *bb++;
  for (i = 0, bb = source->km_HiKeyMapTypes; i < HI_TYPE_SIZE; i++)
    dest->km_HiKeyMapTypes[i] = *bb++;

  for (i = 0, bb = source->km_LoCapsable; i < LO_CAPS_SIZE; i++)
    dest->km_LoCapsable[i] = *bb++;
  for (i = 0, bb = source->km_HiCapsable; i < HI_CAPS_SIZE; i++)
    dest->km_HiCapsable[i] = *bb++;

  for (i = 0, bb = source->km_LoRepeatable; i < LO_REPS_SIZE; i++)
    dest->km_LoRepeatable[i] = *bb++;
  for (i = 0, bb = source->km_HiRepeatable; i < HI_REPS_SIZE; i++)
    dest->km_HiRepeatable[i] = *bb++;
}
```

AppMap_demo.c

```
;/* appmap_demo.c - Execute me to compile me with SAS C 6.2
sc data=far nominc strmer streq nostkchk saveds ign=73 appmap_demo.c
slink FROM LIB:c.o,appmap_demo.o,appkeymap.o TO appmap_demo LIB
LIB:SC.lib,LIB:Amiga.lib
quit
*/

/* Appmap_demo uses the routines in appkeymap.o to clone the current  */
/* task’s console keymap.  Appmap_demo then modifies the keymap clone */
/* by redefining some of the keypad keys.  It installs a string key,  */
/* a "normal" key, and a NO OP key, which are acceptable replacements */
/* for keypad keys on a console-based application.                    */
/*                                                                    */
/* This example also installs a dead/deadable key pair and a double   */
/* dead/deadable key pair.  These are provided only as a means for    */
/* understanding how dead-class keys work.  Do not use them as part   */
/* of anything for public consumption.                                */

#include      <exec/types.h>
#include      <exec/memory.h>
#include      <exec/io.h>
#include      <dos/dos.h>
#include      <devices/console.h>
#include      <devices/keymap.h>

#include      <clib/exec_protos.h>
#include      <clib/dos_protos.h>

#include      <stdio.h>
#include      <stdlib.h>

#ifdef  __SASC
void __regargs __chkabort(void);
void __regargs __chkabort(void){}
#endif

#include "appkeymap.h"

#define MYKEYCOUNT 7

far UBYTE deadmap[] =           /* The data for the dead key (the introducer) */
{
  0, ’4’,                       /* maps to ’4’ when keypad 4 is pressed alone */
  DPF_DEAD, 1                   /* maps to a dead key when shift is down      */
};

far UBYTE deadablemap[] =
{
  0, ’5’,                       /* maps to ’5’ when keypad 5 is pressed alone */
  DPF_MOD, 4,                   /* maps to a deadable key when shift is down  */
  ’5’, 0xC2, ’5’, ’5’, ’5’      /* Entry 1 in this table maps to 0xC2, the    */
};                              /* others map to ’5’.                         */

far UBYTE doubledeadmap[] =
{
  0, ’6’,                              /* maps to ’6’ when keypad 6 is pressed alone */
  DPF_DEAD, 1 | (2 << DP_2DFACSHIFT)   /* maps to a dead key when shift is down      */
};

far UBYTE doubledeadablemap[] =
{
  0, ’7’,                       /* maps to ’7’ when keypad 7 is pressed alone */
  DPF_MOD, 4,                   /* maps to a deadable key when shift is down  */
  ’7’, ’7’, 0xA5, 0xA9, ’7’     /* Entry 2 in this table maps to $A5, entry 3 */
};                              /* to $A9, the others to ’7’.                 */


UBYTE strings[] =
{
        5,16,
        5,21,
        3,26,
        9,29,
        4,38,
        10,42,
        8,52,
        14,60,
        ’p’,’l’,’a’,’i’,’n’,
        ’s’,’h’,’i’,’f’,’t’,
        ’a’,’l’,’t’,
        ’s’,’h’,’i’,’f’,’t’,’+’,’a’,’l’,’t’,
        ’c’,’t’,’r’,’l’,
        ’c’,’t’,’r’,’l’,’+’,’s’,’h’,’i’,’f’,’t’,
        ’c’,’t’,’r’,’l’,’+’,’a’,’l’,’t’,
        ’c’,’t’,’r’,’l’,’+’,’a’,’l’,’t’,’+’,’s’,’h’,’i’,’f’,’t’
};


struct MyKey mykeys[MYKEYCOUNT] =
{
  { N1_KEY,                   /* The ’1’ from the numeric key pad                */
    KCF_STRING | KC_VANILLA,  /* This is a string key that accepts all qualifiers */
    1, 1,                     /* Capsable and Repeatable                         */
    (ULONG) strings           /* This points to the key’s string data table      */
  },


  { N2_KEY,     /* The ’2’ from the numeric key pad                          */
    KC_VANILLA, /* VANILLA key generating shift-alted, alted, shifted, plain  */
    0, 0,       /* Non-Capsable, Non-Repeatable                              */
    0x26252423  /* This long word is four ANSI codes: $26=& $25=% $24=$ $23=# */
  },


  { N3_KEY,     /* The ’3’ from the numeric key pad */
    KCF_NOP,    /* This key is a NO OPeration key   */
    0, 0,       /* Non-Capsable, Non-Repeatable     */
    0L
  },


  { N4_KEY,               /* The ’4’ from the numeric key pad */
    KCF_DEAD | KCF_SHIFT, /* A shiftable Dead-class key       */
    0, 0,                 /* Non-Capsable, Non-Repeatable     */
    (ULONG) deadmap       /* pointer to dead-class key data   */
  },


  { N5_KEY,               /* The ’5’ from the numeric key pad */
    KCF_DEAD | KCF_SHIFT, /* A shiftable Dead-class key       */
    0, 0,                 /* Non-Capsable, Non-Repeatable     */
    (ULONG) deadablemap   /* pointer to dead-class key data   */
  },


  { N6_KEY,               /* The ’6’ from the numeric key pad */
    KCF_DEAD | KCF_SHIFT, /* A shiftable Dead-class key       */
    0, 0,                 /* Non-Capsable, Non-Repeatable     */
    (ULONG) doubledeadmap /* pointer to dead-class key data   */
  },


  { N7_KEY,                   /* The ’7’ from the numeric key pad */
    KCF_DEAD | KCF_SHIFT,     /* A shiftable Dead-class key       */
    0, 0,                     /* Non-Capsable, Non-Repeatable     */
    (ULONG) doubledeadablemap /* pointer to dead-class key data   */
  }
};


extern struct Library *SysBase;

struct KeyMap *appkeymap, defkeymap;

struct IOStdReq *conio = NULL;
struct MsgPort *replyport = NULL;
```

```
/* prototypes for our program functions */

void closeall (void);
void closeout (UBYTE * errstring, LONG rc);
struct IOStdReq *makeio (void);
void freeio (struct IOStdReq *ior);


void
main (int argc, char **argv)
{
  LONG rc = 0;

/* This example is shell-only. It modifies a shell's keymap while running. */
/* Alternately, you could modify this demo code to change the keymap       */
/* of the console.device of another CON: window OR a console.device        */
/* unit you've attached to your own Intuition window.  You need the        */
/* device and unit pointers for the console unit you wish to affect.       */

  if (!argc)
    exit (RETURN_FAIL);

  if (SysBase->lib_Version < 37)
    closeout ("Kickstart 2.0 required", RETURN_FAIL);

  conio = makeio ();
  if (conio == NULL)
    closeout ("Can't create IORequest", RETURN_FAIL);


  conio->io_Command = CD_ASKKEYMAP;          /* Obtain a copy of the    */
  conio->io_Data = (APTR) & defkeymap;       /* shell console's KeyMap. */
  conio->io_Length = sizeof (struct KeyMap);
  DoIO (conio);

  appkeymap = CreateAppKeyMap (&defkeymap);

  if (appkeymap == NULL)
    closeout ("Can't create keymap", RETURN_FAIL);

/* If we get here, all went OK.  We now have appkeymap and defkeymap, and  */
/* conio IOStdRequest is init'd to talk to our shell's console device unit */

  AlterAppKeyMap (appkeymap, mykeys, MYKEYCOUNT);

  conio->io_Command = CD_SETKEYMAP;      /* Set the keymap for the console */
  conio->io_Data = (APTR) appkeymap;     /* to be the modified clone.      */
  conio->io_Length = sizeof (struct KeyMap);
  DoIO (conio);

  printf ("Appkeymap installed for this console.\n");
  printf ("Changes for our keymap were specified in an array in appmap_demo.c\n\n");
  printf ("Numeric pad 1 is now a repeatable and capsable string key.  It\n");
  printf ("    also prints a different string for all qualifier combos.\n");
  printf ("Numeric pad 2 is different chars if normal, shifted, alted, shift-
alted.\n");
  printf ("Numeric pad 3 is disabled.\n");
  printf ("Shift-Numeric pad 4 is a dead-key.  Shift-Numeric pad 5 is a deadable
key.\n");
  printf ("    When you hit Shift-Numeric pad 4 then Shift-Numeric pad 5, this\n");
  printf ("    example prints '\302'\n");
  printf ("Shift-Numeric pad 6 is a double dead-key.  Shift-Numeric pad 7 is a
deadable\n");
  printf ("    key.  When you hit Shift-Numeric pad 6 once then Shift-Numeric pad
7,\n");
  printf ("    this example prints '\245'.  When you hit Shift-Numeric pad 6 twice
then\n");
  printf ("    Shift-Numeric pad 7, this example prints '\251'.\n");
  printf ("\nHit return to exit when done\n");
  getchar ();
  printf ("Setting this console back to default keymap\n");

  conio->io_Command = CD_SETKEYMAP;          /* Restore the shell's original keymap */
  conio->io_Data = (APTR) & defkeymap;
  conio->io_Length = sizeof (struct KeyMap);
  DoIO (conio);

  closeall ();
  exit (rc);
}



struct IOStdReq *
makeio (void)
{
  struct MsgPort *conport;
  struct IOStdReq *ior = NULL;
  struct InfoData *id;
  struct Process *proc;

  proc = (struct Process *) FindTask (NULL);
  conport = (struct MsgPort *) proc->pr_ConsoleTask;
  if (!conport)
    return (NULL);

  if (id = (struct InfoData *)
      AllocMem (sizeof (struct InfoData), MEMF_PUBLIC | MEMF_CLEAR))
  {
    if (DoPkt (conport, ACTION_DISK_INFO, ((ULONG) id) >> 2, 0L, 0L, 0L, 0L))
    {
      if (replyport = CreateMsgPort ())
      {
        if (ior = CreateIORequest (replyport, sizeof (struct IOStdReq)))
        {
          ior->io_Device = ((struct IOStdReq *) id->id_InUse)->io_Device;
          ior->io_Unit = ((struct IOStdReq *) id->id_InUse)->io_Unit;
        }
      }
    }
    FreeMem (id, sizeof (struct InfoData));
  }
  return (ior);
}

void
freeio (struct IOStdReq *ior)
{
  if (ior)
  {
    if (ior->io_Message.mn_ReplyPort)
      DeleteMsgPort (ior->io_Message.mn_ReplyPort);
    DeleteIORequest (ior);
  }
}

void
closeall ()
{
  if (conio)
    freeio (conio);
  if (appkeymap)
    DeleteAppKeyMap (appkeymap);
}


void
closeout (UBYTE * errstring, LONG rc)
{
  if (*errstring)
    printf ("%s\n", errstring);
  closeall ();
  exit (rc);
}
```